RTCU Communication Support Package

Version 4.50

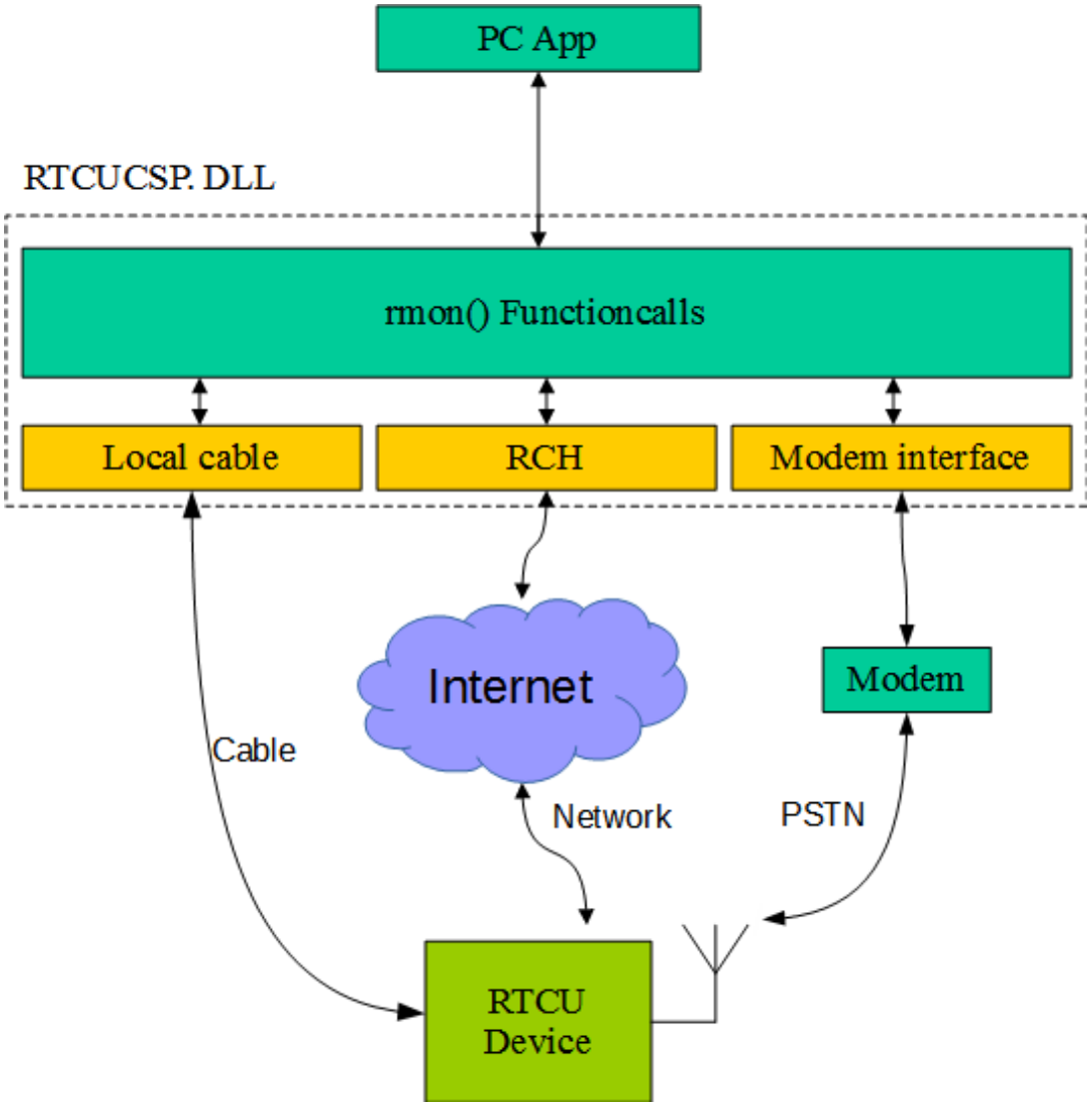


Table of Contents

Introduction

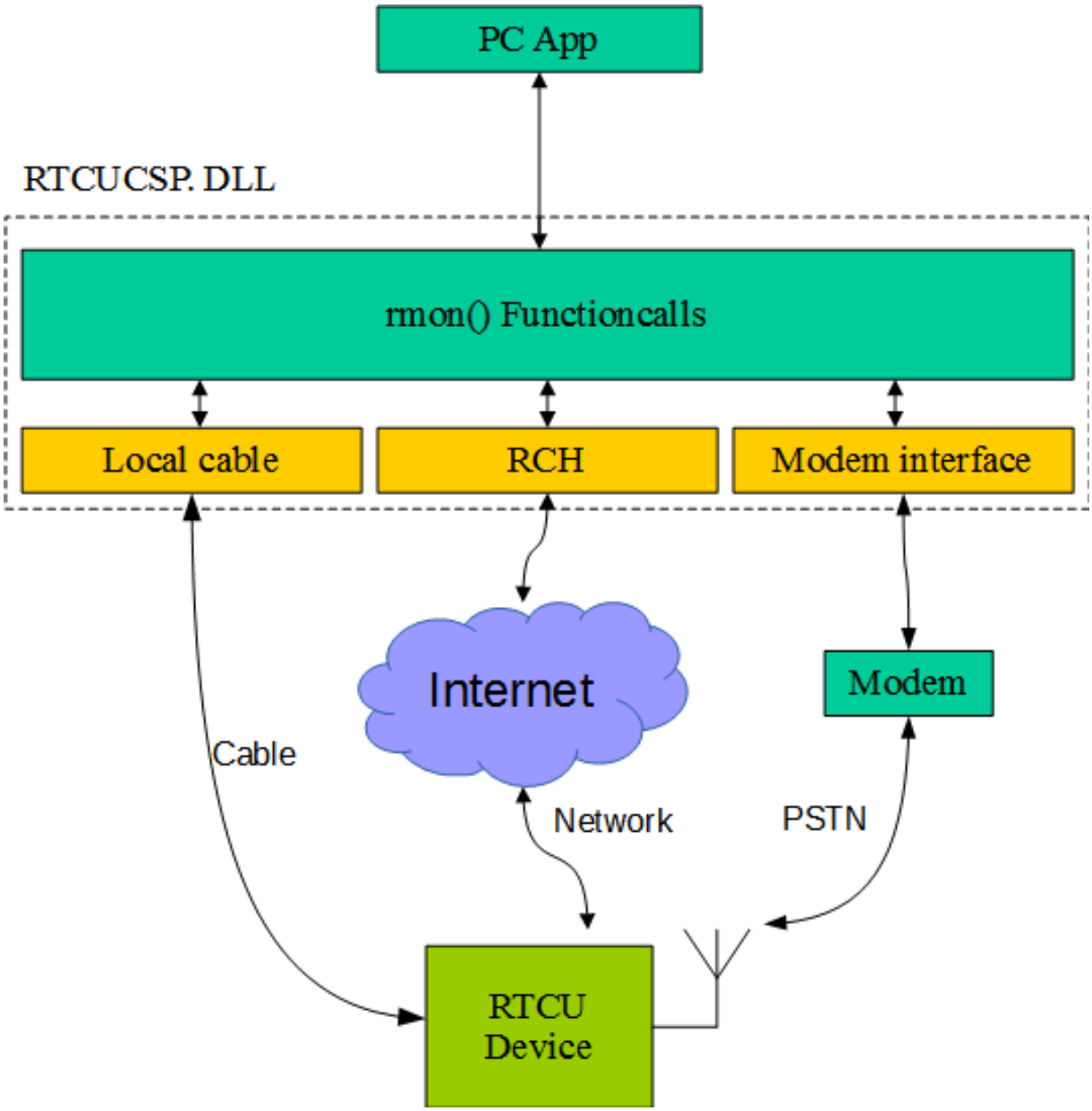
This document describes the RTCU CSP (Communication Support Package), an API library of functions that allows communication with RTCU products on the same level of functionality available when using the RTCU IDE.

Establishing a connection to the RTCU device can be done over a direct cable connection or remotely over the RTCU Communication Hub (RC. CSD (modem) is also supported for devices that support this older connection technology.

For more information on the RACP protocols used by the CSP, please refer to the: "RTCU Communication Protocol Documentation Set."

This document also contains the source code for a simple application (Appendix A) and a complete application with the RTCU Programming Tool application (Appendix B).
The RTCU Programming Tool is a full-featured application that can be downloaded as a fully installable package that allows uploading applications and firmware as well as setting up various device parameters.
The RTCU Programming Tool shows all the different aspects of establishing and maintaining an application with an RTCU device, authentication, etc.
Both applications will give an excellent hands-on experience to the library and function as a good starting point for application development.

Graphic illustration of the library



Contents of package

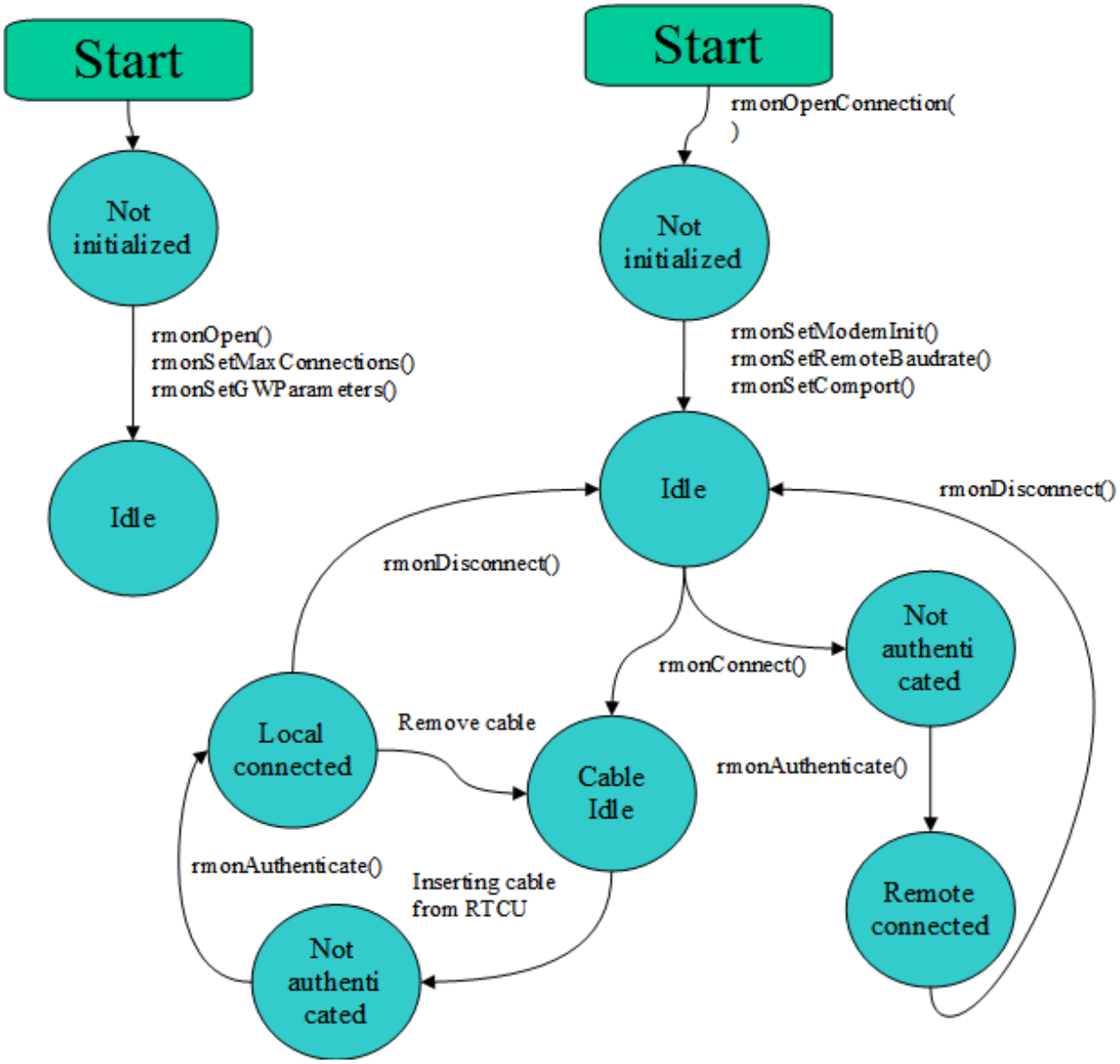
The package this document is part of, contains the following:

"\RTCU Communication Support Package.pdf"	This document
"\RTCUPROG\"	The RTCUPROG application described in appendix B.
"\Demo"	A simple 32/64 bit Demo application.
"\Library\"	The library root folder.
"\Library\H"	Library header file folder.
"\Library\LIB32"	Library for 32-bit applications.
"\Library\LIB64"	Library for 64-bit applications.
"\Library\DLL32"	DLL files for 32-bit applications.
"\Library\DLL64"	DLL files for 64-bit applications.

Microsoft Visual Studio C++ 2019 will be needed to use and build this library.

Interface and state diagram

To be able to go into details about using the interface, it is necessary to know the different states in which the communication protocol can be:



Initializing the library

To begin with, you are in the **Not Initialized State**. To proceed from there, you will have to carry out the following two steps:

Step A.

You will have to **prepare the communication system** with rmonSetGWParameters and rmonSetMazConnections

Step B.

You will have to **open the communication system** with rmonOpen().

Entering the Idle State.

After the communication system is initialized, the **Idle State** is entered.
From there, it is possible to open connections to RTCU devices.

Initializing the connection.

After calling the rmonOpenConnection, you are in the **Not Initialized State**. To proceed from there, you will have to carry out the following two steps:

Step A.

Then determine **what kind of connection** you want to be started.
Should it be a local cable connection?
Should it be a data call (CSD) modem connection?
Should it be an RTCU Communication Hub connection?

Step B.

You will have to **prepare the connection** with rmonSetComport(), rmonSetModemInit() and rmonSetRemoteBaudrate() if connection is local cable or CSD modem.

Entering the connection Idle State.

After the connection is initialized, the **Connection Idle State** is entered.
From there it is possible to connect to an RTCU device either thru
A local cable connection or
A remote connection – (modem (CSD) or RTCU Communication Hub connection)

For remote connections calling rmonConnect() will enter the **Not Authenticated State**.
For cable connection, calling rmonConnected will enter the **Cable Idle State**, and from here, inserting a cable between the PC and RTCU device will enter the **Not Authenticated State**.

From the **Not Authenticated State** carry out the rmonAuthenticate(), which will put the library into either the **Locally Connected State** or the **Remotely Connected State.**

The Local/Remote Connected State.

The communication is now up running, and your PC application can now communicate with the RTCU device using the functions described.

Closing or changing the connection.

A. The Locally Connected State:

As you can see in the state diagram above, you have **three possibilities of** being in a **Locally Connected State.** When you want to leave this state, you can either

Remove the cable, which will bring you into the **Cable Idle State**.
Use rmonDisconnect(), which will bring you into the **Connection Idle State**.
Close the connection (using rmonCloseConnection).

B. The Remotely Connected State:
As you can see in the state diagram above, you have **three possibilities of** being in a **Remotely Connected State.** When you want to leave this state, you can either
Use rmonDisconnect(), which will bring you into the **Connection Idle State**.
Close the connection (using rmonCloseConnection).

Functions in the RTCUCSP.DLL library

In the following description of the different function calls, we will differentiate between NX32L, NX32 and X32 RTCU devices, as some of the functions are not supported on some models.
Please consult Logic IO's website for up-to-date information on new products and their availability.

Return codes

The return codes from most of the functions will be one of those shown below. These return codes are declared in the header file (rtcucsp.h) for the library as an "enum struct rmonRet".

Symbolic name	Value	Description
rmonOK	0	Operation OK
rmonError	1	General error
rmonComError	2	Communication error
rmonTargetError	3	Other error in device than rmonComError
rmonIllegalHandle	4	The connection handle is illegal (corrupt/non existent)
rmonIllegalTarget	5	Target and type of firmware file does not match
rmonOnlyGateway	6	The function can only be used via the RCH
rmonNotGateway	7	The function can not be used over the RCH
rmonDenied	8	Access to device denied
rmonNoData	9	No data
rmonNoMoreData	10	No more data
rmonNotInit	11	Not initialized
rmonInit	12	The RTCUCSP library is already initialized
rmonConnection	13	There is a connection established already
rmonGatewayNotFound	14	RCH not found
rmonFileNotFound	15	Application or Firmware file not found.
rmonIllegalFile	16	File specified is illegal (corrupt)
rmonOldFormat	17	Old firmware file format, not supported
rmonNoMonitorMode	18	Not able to enter monitor mode
rmonErrorReset	19	Not able to reset RTCU
rmonErrorHalt	20	Not able to halt RTCU
rmonNoBackground	21	Background transfer not supported by the RTCU device.
rmonInterrupted	22	Background transfer was interrupted.
rmonCancelled	23	Data transfer has been canceled.
rmonNotProgrammable	24	Attempt to program a Micro device with a VSX file
rmonNoModem	25	No remote serial port selected or port in use.
rmonNoCable	26	No local serial port selected or port in use.
rmonMemoryConfig	27	Memory configuration is different in Project and RTCU device.
rmonImageTooLarge	28	There are no room for the image in the RTCU device.
rmonNotModem	29	The function can not be used over Modem connection.
rmonIllegalAccess	30	The function is not accessible.
rmonDowngrade	31	The device does not support this firmware version
rmonImageSupport	33	The RTCU device does not support the image.

Initialization/Configuration

Functions that are usually needed before any real communication can be started with the RTCU device are listed here. For the proper sequence of calling the functions, please refer to the State diagram, and to the demo applications delivered as part of this package (see appendix A and appendix B).

rmonOpen ()	**RTCU architecture:** n/a **Called in:** Not Initialised
Synopsis	rmonRet __stdcall rmonOpen(void)
Description	Opens and initializes the communication system. The system can be closed again by calling rmonClose().
Returns	rmonOK, rmonError, rmonInit

rmonClose()	**RTCU architecture:** n/a **Called in:** Idle State
Synopsis	rmonRet __stdcall rmonClose(void)
Description	Closes communication system. The communication system must be opened with rmonOpen() in order for it to be used again.
Returns	rmonOK, rmonNotInit

rmonGetVer()	**RTCU architecture:** n/a **Called in:** Not Initialised and Idle State
Synopsis	int __stdcall rmonGetVer(void)
Description	Retrieve the version number of the RTCUCSP library.
Returns	Library version scaled by 100.

rmonSetMaxConnections()	**RTCU architecture:** n/a **Called in:** Not Initialised
Synopsis	rmonRet __stdcall rmonSetMaxConnections (int max_sessions)
Description	Set maximum number of simultaneous connections possible. If this function is not called the default number of simultaneous connections will be used.
Input	**max_sessions** — The maximum number of connections. (Default: 100)
Returns	rmonOK, rmonInit

rmonSetGWParameters()

RTCU architecture: n/a
Called in: Not Initialised

Synopsis

rmonRet __stdcall rmonSetGWParameters(const unsigned short Port, const unsigned long MyNodeID, const char* IP, const char* Key)

Description

If a connection to a remote RTCU is to be done using the RTCU Communication Hub, some parameters have to be set before this is possible. These parameters are set using this function. Please see the online help to the RTCU IDE for a description of these parameters (Menu: Device - > Connection -> RTCU Communication Hub).

Input

Port	RCH port (please refer to rmonConnect() for making a connection thru the RCH)
MyNodeID	Node ID (can be set to 0, this will allow the RCH to issue a ”dynamic” ID to this node).
IP	IP address of the RCH.
Key	Key value (must be set to the same value as set in the RCH)

Returns

rmonOK, rmonInit

rmonSetGWParametersAdv()

RTCU architecture: n/a
Called in: Not Initialised

Synopsis

rmonRet __stdcall rmonSetGWParametersAdv(unsigned char CryptKey[16], unsigned char gw_max_connection_attempt, unsigned char gw_max_send_req_attempt, unsigned short gw_response_timeout, unsigned short gw_alive_freq)

Description

This function is used to set the advanced parameters for connecting to a remote RTCU device using the RCH
Please see the online help to the RTCU IDE for a description of these parameters (Menu: Device - > Connection -> RTCU Communication Hub).

Input

CryptKey	Encryption key. To use the default key, set all 16 bytes to 0 (zero).
gw_max_connection_attempt	Maximum connection attempts. Default value: 3. Range: 1-60
gw_max_send_req_attempt	Maximum transmission attempts. Default value: 3. Range: 1-60
gw_response_timeout	Response timeout. Default value: 30. Range: 5-60
gw_alive_freq	Keep alive frequency. Default value: 60. Range: 0-60000

Returns

rmonOK, rmonInit

rmonSetRCHParametersSecure()

RTCU architecture: n/a
Called in: Not Initialised

Synopsis
rmonRet __stdcall rmonSetRCHParametersSecure(char* server, char* cert, char* key, char* password)

Description
This function is used to set the parameters for making a secure connecting to a remote RTCU device using the RTCU Communication Hub.
Please see the online help to the RTCU IDE for a description of these parameters (Menu: Device - > Connection -> RTCU Communication Hub).

Input

Server	The location of the root CA certificate to validate the server certificate.
Cert	The location of the client certificate of the CSP.
Key	The location of the private key of the client certificate.
Password	The password for the private key.

Returns
rmonOK, rmonInit, rmonError

rmonGetPortList()

RTCU architecture: n/a
Called in: Not Initialised and Idle State

Synopsis
rmonRet __stdcall rmonGetPortList (char name[RMON_MAXPORTS][9], int* size)

Description
Retrieve the names and number of comports available at start up.

Output

Name	Array of ASCIIZ strings with the names of the comports.
size	The number of comports present.

Returns
rmonOK, rmonNoData

rmonEnumeratePorts()

RTCU architecture: n/a
Called in: All states

Synopsis
rmonRet __stdcall rmonEnumeratePorts (rmoncbserialport cbFunc, void *arg, int *count)

Description
Enumerate the serial ports currently present.

Input

cbFunc	Function that is called with the name and description of a serial port.
arg	A user defined argument that is included in the callback function.

Output

count	The number of serial ports enumerated.

Returns
rmonOK

The call-back function is defined as follows:
typedef void (__stdcall *rmoncbserialport)(void *arg, const char *name, const char *desc);

rmonOpenConnection()	**RTCU architecture:** n/a **Called in:** Idle State
Synopsis	HRMONCON __stdcall rmonOpenConnection (void)
Description	Open a new connection session.
Returns	Handle to connection or HRMONCON_ILLEGAL if no connection.

rmonCloseConnection()	**RTCU architecture:** n/a **Called in:** Not Initialised and Idle State
Synopsis	rmonRet __stdcall rmonCloseConnection (HRMONCON hCon)
Description	Close a connection session.
Input	**hCon** Handle to connection
Returns	rmonOK, rmonIllegalHandle

rmonSetComport()	**RTCU architecture:** n/a **Called in:** Not Initialised and Idle State
Synopsis	rmonRet __stdcall rmonSetComport(HRMONCON hCon, const char* LocalPort, const char* RemotePort)
Description	The connection needs to know which serial ports on the PC is going to be used for communication, both for direct cable connection, and for connection thru a modem. These ports are configured using this call. If one of the ports is not to be used, simply set it to “COM0”. To configure an USB cable connection, use the port names “USB1” thru “USB8”.
Input	**hCon** Handle to connection. **LocalPort** Name of the COM port to be used for direct cable connection. **RemotePort** Name of the COM port to be used for remote connection thru a modem.
Returns	rmonOK, rmonConnection, rmonIllegalHandle

rmonSetModemInit()	**RTCU architecture:** n/a **Called in:** Not Initialised and Idle State

Synopsis rmonRet __stdcall rmonSetModemInit(HRMONCON hCon, const char* atcmd)

Description Specifies initialisation string to modem for usage in remote connection. A list of common initialization string for different types of modems can be seen in the RTCU IDE (menu: Settings -> Setup).

Input

hCon	Handle to the connection.
Atcmd	Initialisation string for modem

Returns rmonOK, rmonConnection, rmonIllegalHandle

rmonSetRemoteBaudrate()	**RTCU architecture:** n/a **Called in:** Not Initialised and Idle State

Synopsis rmonRet __stdcall rmonSetRemoteBaudrate(HRMONCON hCon, int baud)

Description This function sets the baud rate that will be used when communication is established to a remote device thru a modem. This is the baud rate used between the PC and the Modem, and has nothing to do with the speed being used on the link between the modem and RTCU device (which can be influenced by setting the appropriate settings in the rmpnSetModemInit()).

Input

hCon	Handle to connection
Baud	Baud rate to be used. If baud is set to 0 a default value of 57600 baud is used. If allowed by hardware the baud rate in principle can be set to anything Commonly used baud rates are: 9600, 19200, 38400, 57600 and 115200 Other protocol parameters are: No parity, 8 data bits and 1 stop bit.

Returns rmonOK, rmonConnection, rmonIllegalHandle

rmonConnect()	**RTCU architecture:** n/a **Called in:** Idle

Synopsis rmonRet __stdcall rmonConnect(HRMONCON hCon, const char* phonenumber)

Description This function is used to establish a connection to a RTCU device. The connection can be either over cable, thru a modem, or thru the RTCU Communication Hub. If the remote RTCU is to be contacted thru a modem, simply use the telephone number of the SIM card in the RTCU, if connection is thru the RTCU Communication Hub, the devices serial number (nodeid) is to be used, prefixed with a "@" character! To establish a connection over cable leave the phone number empty.

Input

hCon	Handle to connection
phonenumber	The phone number of the SIM card in the remote RTCU if a connection is thru modem, or the serial number of the RTCU (prefixed with "@") if connection is thru RCH.

Returns rmonOK, rmonIllegalHandle, rmonConnection, rmonDenied, rmonComError, rmonError

rmonDisconnect()

RTCU architecture: n/a
Called in: Remotely Connected State

Synopsis rmonRet __stdcall rmonDisconnect(HRMONCON hCon)

Description Disconnect a connection to a RTCU device.

Input

hCon	Handle to connection

Returns rmonOK, rmonIllegalHandle, rmonError

rmonConnected()

RTCU architecture: n/a
Called in: After Initialised State

Synopsis rmonRet __stdcall rmonConnected(HRMONCON hCon)

Description rmonConnected() returns the type of connection that is currently (if any) established with the RTCU device.

Input

hCon	Handle to connection

Returns Type of connection:

Symbolic name	Value	Description
RMONCON_NONE	0	Currently not connected
RMONCON_LOCAL	1	Connected using cable
RMONCON_REMOTE	2	Connected thru a modem
RMONCON_GW	3	Connected thru the RCH

rmonAuthenticate()

RTCU architecture: All
Called in: Not Authenticated State

Synopsis rmonRet __stdcall rmonAuthenticate (HRMONCON hCon, const char password[21])

Description if there is established a (new) connection with a RTCU device, either local or remote, the first thing to do, is for the application to authenticate itself for the RTCU device. This is done using this function, and must be done, BEFORE any other communication with the device can take place.

If there is no password set in the RTCU device, this function must still be called, just with an empty string “” as the password.

Input

hCon	Handle to connection
password	Password, zero terminated ASCII string

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonIllegalTarget, rmonTargetError, rmonDenied

rmonEnableLargeData()

RTCU architecture: All
Called in: Not Initialised and Idle State

Synopsis

rmonRet __stdcall rmonEnableLargeData(HRMONCON hCon, unsigned char enable)

Description

This function enables the support for larger data frames on medias that supports them. This allows for transferring more data for each call to some functions, including rmonLogReadExt and rmonLogReadByTag.

Input

hCon	Handle to connection
enable	0=disable, 1=enable

Returns

rmonOK, rmonIllegalHandle

Program/Firmware upload

The following functions are used for uploading new applications, voice messages and firmware to a RTCU device:
All functions reports their progress, by calling an optional call-back function, defined as follows:

typedef int (__stdcall *rmoncbprogress)(void* uptr,int percent);

Please note that if the call-back function returns a value different from 0 (zero) the functions will cancel.

rmonFirmwareUpload()	**RTCU architecture:** All **Called in:** Locally Connected State

Synopsis

rmonRet __stdcall rmonFirmwareUpload(HRMONCON hCon, char *FirmwareFilename, rmoncbprogress cbfunc, void *uptr);

Description

Upload firmware to RTCU.
The function takes a .BIN firmware file, and transfers it to an RTCU device. The function will halt execution in the device, upload the new firmware file, and after the transfer, it will reset the device. Note that to upload a new firmware to an RTCU device when the connection to it is via the RCH, you need to use background update (see rmonFirmwareStartUpload).

Input

hCon	Handle to connection
FirmwareFilename	Firmware file
cbfunc	Call-back function for progress
uptr	User data that will be passed to call-back function when called

Returns

rmonOK, rmonComError, rmonIllegalHandle, rmonIllegalTarget, rmonNotGateway, rmonNoMonitorMode, rmonErrorReset, rmonCancelled, rmonFileNotFound, rmonIllegalFile, rmonOldFormat, rmonNotModem, rmonDowngrade

rmonFirmwareStartUpload()

RTCU architecture: All
Called in: Connected State

Synopsis

rmonRet __stdcall rmonFirmwareStartUpload(HRMONCON hCon, const char* Filename, struct rmonBGReport* Report, rmoncbprogress cbfunc, void* uptr);

Description

Upload firmware to RTCU.

Function takes a .BIN firmware file, and transfers it to a RTCU device. Unlike rmonFirmwareUpload the function will not halt execution in the device, but start to upload the firmware in the background. The upload started with this function supports resume if the upload is interrupted. If the upload is interrupted the Report structure will contain the information needed to resume the upload using rmonFirmwareResumeUpload.
The newly uploaded firmware is used after the device has been reset.
Note that part of the voice memory in the X32 device is used for the update, and any voice data must be uploaded again. Use the function rmonVoiceMessagesAbove64K to determine if the use of this function will overwrite any voice data.

Input

hCon	Handle to connection
Filename	Zero terminated string with the firmware filename
Report	A structure containing progress status (see definition below)
cbfunc	Call-back function for progress
uptr	User data that will be passed to call-back function when called

Returns

rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError, rmonCancelled, rmonNoBackground, rmonFileNotFound, rmonOldFormat, rmonIllegalFile, rmonIllegalTarget, rmonInterrupted, rmonDowngrade

```
struct rmonBGReport {
   unsigned long  extSeg;      // External segment written
   unsigned long  intSeg;      // Internal segment written
   unsigned long  headSeg;     // Header segment written
};
```

rmonFirmwareResumeUpload()

RTCU architecture: All
Called in: Connected State

Synopsis

rmonRet __stdcall rmonFirmwareResumeUpload(HRMONCON hCon, const char* Filename, struct rmonBGReport* Report, rmoncbprogress cbfunc, void* uptr);

Description

Upload firmware to RTCU.

Function takes a report containing a .BIN firmware file and progress status, and resumes where the upload was interrupted. If the upload is interrupted the Report structure will contain the information needed to resume the upload.
Note that this function cannot be used to start a new upload, only complete an interrupted upload.
Note that part of the voice memory in the X32 device is used for the update and voice data must be uploaded again.

Input

hCon	Handle to connection
Filename	Zero terminated string with the firmware filename
Report	A structure containing progress status (see definition below)
cbfunc	Call-back function for progress
uptr	User data that will be passed to call-back function when called

Returns

rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError, rmonCancelled, rmonNoBackground, rmonFileNotFound, rmonOldFormat, rmonIllegalFile, rmonIllegalTarget, rmonInterrupted

```
struct rmonBGReport{
   unsigned long  extSeg;      // External segment written
   unsigned long  intSeg;      // Internal segment written
   unsigned long  headSeg;     // Header segment written
};
```

rmonApplicationUpload()

RTCU architecture: All
Called in: Connected State

Synopsis

rmonRet __stdcall rmonApplicationUpload(HRMONCON hCon, char *Filename, rmoncbprogress cbfunc, void *uptr);

Description

Uploads application to RTCU
Function takes a .VSX, a .PSX or a .RPC file, and transfers it to a RTCU.
Please note that the execution of the VPL program in the RTCU device **must** be halted with rmonHalt(), **before** calling this function ! The RTCU will have to be reset after the transfer, to start the new application.

Input

hCon	Handle to connection
Filename	Zero terminated string with the filename of the application
cbfunc	Call back function for progress
uptr	User data that will be passed to call back function when called

Returns

rmonOK, rmonComError, rmonIllegalHandle, rmonErrorHalt, rmonFileNotFound, rmonIllegalFile, rmonNotProgrammable, rmonCancelled, rmonTargetError, rmonIllegalTarget, rmonImageTooLarge, rmonImageSupport

rmonApplicationStartUpload()

RTCU architecture: All
Called in: Connected State

Synopsis

rmonRet __stdcall rmonApplicationStartUpload(HRMONCON hCon, const char* Filename, struct rmonBGReport* Report, rmoncbprogress cbfunc, void *uptr);

Description

Uploads application to RTCU
Function takes a Report containing a .VSX, a .PSX or a .RPC file, and transfers the file to a RTCU. The upload will be performed in the background without interfering with the running application. The upload started with this function supports resume if the upload is interrupted. If the upload is interrupted the Report structure will contain the information needed to resume the upload using rmonApplicationResumeUpload.
The newly uploaded application is used after the device has been reset.
Note that the voice memory in the X32 device is used, and any voice data must be uploaded again. Use the function rmonVoiceMessagesAbove64K to determine if the use of this function will overwrite any voice data.

Input

hCon	Handle to connection
Filename	Zero terminated string with the filename of the application
Report	A structure containing progress status (see definition below)
cbfunc	Call back function for progress
uptr	User data that will be passed to call back function when called

Returns

rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError, rmonIllegalTarget, rmonCancelled, rmonNoBackground, rmonFileNotFound, rmonIllegalFile, rmonNotProgrammable, rmonInterrupted, rmonImageTooLarge, rmonImageSupport

```
struct rmonBGReport {
   unsigned long  extSeg;      // External segment written
   unsigned long  intSeg;      // Internal segment written
   unsigned long  headSeg;     // Header segment written
};
```

rmonApplicationResumeUpload()

RTCU architecture: All
Called in: Connected State

Synopsis

rmonRet __stdcall rmonApplicationResumeUpload(HRMONCON hCon, const char* Filename, struct rmonBGReport* Report, rmoncbprogress cbfunc, void *uptr);

Description

Uploads application to RTCU
Function takes a report containing a .VSX, a .PSX or a .RPC file and progress status, and resumes where the upload was interrupted. If the upload is interrupted the Report structure will contain the information needed to resume the upload.
Note that this function cannot be used to start a new upload, only complete an interrupted upload.
Note that the voice memory in the X32 device is used, and any voice data must be uploaded again.

Input

hCon	Handle to connection
Filename	Zero terminated string with the filename of the application
Report	A structure containing progress status (see definition below)
cbfunc	Call back function for progress
uptr	User data that will be passed to call back function when called

Returns

rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError, rmonIllegalTarget, rmonCancelled, rmonNoBackground, rmonFileNotFound, rmonIllegalFile, rmonNotProgrammable, rmonInterrupted, rmonImageTooLarge, rmonImageSupport

```
struct rmonBGReport {
   unsigned long  extSeg;      // External segment written
   unsigned long  intSeg;      // Internal segment written
   unsigned long  headSeg;     // Header segment written
};
```

rmonVoiceUpload()

RTCU architecture: X32 & NX32
Called in: Connected State

Synopsis

rmonRet __stdcall rmonVoiceUpload(HRMONCON hCon, char *ProjectFilename, rmoncbprogress cbfunc, void *uptr);

Description

Upload Voice messages to RTCU.
Function transfers all voice messages associated with a RTCU project.

It is important that the relative directory structure for the project is maintained as when the project was built in the RTCU IDE environment, otherwise the function will have trouble locating the voice message files.

Please note that the execution of the VPL program in the RTCU device must be halted with rmonHalt(), before calling this function ! The RTCU will have to be reset after the transfer for the new voice messages to take effect.

Input

hCon	Handle to connection
ProjectFilename	Project filename
cbfunc	Call back function for progress
uptr	User data that will be passed to call back function when called

Returns

rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError, rmonCancelled, rmonFileNotFound, rmonIllegalFile, rmonMemoryConfig, rmonImageTooLarge

rmonNumOfVoiceMessages()

RTCU architecture: n/a
Called in: All states

Synopsis

rmonRet __stdcall rmonNumOfVoiceMessages(char *ProjectFilename, int *NumFiles);

Description

Determine how many voice messages is included in a project. This is useful for determining if the rmonVoiceUpload() function has to be called when uploading a complete project to a RTCU device.

Input

ProjectFilename	**Project filename**
NumFile	Number of voice file in PRJ file

Returns

rmonOK, rmonFileNotFound, rmonIllegalFile

rmonCheckTransfer()

RTCU architecture: All
Called in: Connected State

Synopsis

rmonRet __stdcall rmonCheckTransfer(HRMONCON hCon, int type, const char* filename, int* report);

Description

Determine if there are any special considerations for performing a transfer.

Input

hCon	**Handle to connection.**
Type	The type of transfer to check, see below.
filename	The name of the file to transfer.

Output

report	Report on special conditions, see below.

Returns

rmonOK, rmonFileNotFound, rmonIllegalFile

Type of transfer:

Symbolic name	Value	Description
RMONCT_APP_DIRECT	1	Direct application transfer.
RMONCT_APP_BACKGROUND	2	Background application transfer.
RMONCT_FW_DIRECT	3	Direct firmware transfer.
RMONCT_FW_BACKGROUND	4	Background transfer.
RMONCT_VOICE	5	Voice transfer.

Transfer report:

Symbolic name	Value	Description
RMONCT_REP_VOICE	1	Voice in device will be erased.
RMONCT_REP_FW300	2	Firmware version 3.00 is required.
RMONCT_REP_DENIED	3	Voice transfer is not possible.

rmonApplicationFilename()	**RTCU architecture:** All **Called in:** Connected State

Synopsis

rmonRet __stdcall rmonApplicationFilename(const char* ProjectFilename, char* Filename);

Description

Provided with the name of the project file, this function determines the name of the corresponding application file.

Input

ProjectFilename	ASCIIZ string with the name of the project file with the file extension .PRJ

Output

Filename	Name of application file generated from project. Must be large enough to store a filename with the same length as ProjectFilename.

Returns

rmonOK, rmonFileNotFound, rmonIllegalFile

Manipulation of Persistent memory

The Persistent memory of the RTCU device, can be manipulated with this set of functions. The FLASH based Persistent memory in the RTCU devices, is accessible from VPL using the functions SaveData/LoadData, SaveString/LoadString. The FRAM based memory, is accessible with the functions SaveDataF / LoadDataF, SaveStringF / LoadStringF.

rmonPersistentRead()

RTCU architecture: All
Called in: Connected State

Synopsis

rmonRet __stdcall rmonPersistentRead(HRMONCON hCon, int first, int last, int type, char* data, rmoncbprogress pfunc, void* uptr, int reserved);

Description

Reads a range of entries from Persistent memory with bounds check.

Input

hCon	**Handle to connection**
first	1 based index of first entry to read
last	1 based index of last entry to read
type	The type of persistent memory to read from, see below
pfunc	Pointer to progress callback function
uptr	Pointer to user argument used when reporting progress.
reserved	Reserved for future use. Must be set to zero

Output

data	Buffer to store the data in. The buffer is handled as a packed array of the structure rmonPersistEntry, see below.

Returns

rmonOK, rmonComError, rmonIllegalHandle, rmonError, rmonCancelled, rmonIllegalAccess

type:

Symbolic name	Value	Description
RMON_TYPE_FRAM	1	FRAM
RMON_TYPE_FLASH	2	FLASH
RMON_TYPE_XFLASH	3	Externded FLASH

```
struct rmonPersistEntry {
  unsigned char type;      // The type of entry, see below
  short length;            // The number of bytes in the entry
  unsigned char data[255]; // The contents of the persistent entry
};
```

type:

Symbolic name	Value	Description
RMON_PERSIST_TEXT	1	Text entry
RMON_PERSIST_BINARY	2	Binary entry

rmonPersistentWrite()

RTCU architecture: All
Called in: Connected State

Synopsis

rmonRet __stdcall rmonPersistentRead(HRMONCON hCon, int first, int last, int type, char* data, rmoncbprogress pfunc, void* uptr, int reserved);

Description

Writes a range of entries from Persistent memory with bounds check.

Input

hCon	**Handle to connection**
first	1 based index of first entry to read
last	1 based index of last entry to read
type	The type of persistent memory to write to, see below
data	Buffer with entries to write. The buffer is handled as a packed array of the structure rmonPersistEntry, see below.
pfunc	Pointer to progress callback function
uptr	Pointer to user argument used when reporting progress.
reserved	Reserved for future use. Must be set to zero

Returns

rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError, rmonError, rmonIllegalAccess

type:

Symbolic name	Value	Description
RMON_TYPE_FRAM	1	FRAM
RMON_TYPE_FLASH	2	FLASH
RMON_TYPE_XFLASH	3	Externded FLASH

```
struct rmonPersistEntry {
  unsigned char type;      // The type of entry, see below
  short length;            // The number of bytes in the entry
  unsigned char data[255]; // The contents of the persistent entry
};
```

type:

Symbolic name	Value	Description
RMON_PERSIST_TEXT	1	Text entry
RMON_PERSIST_BINARY	2	Binary entry

rmonReadPersistentFRAM()

RTCU architecture: All
Called in: Connected State

Synopsis

rmonRet __stdcall rmonReadPersistentFRAM(HRMONCON hCon, int entry, char *data, int *length, int binary);

Description

Read FRAM based persistent entry.
This function reads a specific entry from FRAM based Persistent memory in the RTCU. When called, you must specify what type of data you are expecting to read, if the specified type of data is not present, the function returns rmonNoData, otherwise the data and length are returned.
Data read with this function, can be stored from VPL with the functions SaveStringF() and SaveDataF()

Input

hCon	**Handle to connection**
entry	Entry number, from 1 to 20 on X32 and from 1 to 100 on NX32 and NX32L.
binary	Set to 1 if expecting binary data, 0 if string expected

Output

data	**Buffer for data (must be large enough!)**
length	The number of bytes read from the entry

Returns

rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError, rmonNoData, rmonError

rmonWritePersistentFRAM()

RTCU architecture: All
Called in: Connected State

Synopsis

rmonRet __stdcall rmonWritePersistentFRAM(HRMONCON hCon, int entry, char *data, int length, int binary);

Description

Write to FRAM based persistent entry.
This function writes either binary data or a string to a specific entry in the FRAM based persistent memory in the RTCU. When called, you must specify what type of data you are storing.
Data stored with this function, can be read from VPL with the functions LoadStringF() and LoadDataF().

Input

hCon	Handle to connection
entry	Entry number, from 1 to 20 on X32 and from 1 to 100 on NX32 and NX32L.
data	The data to store
length	Length of data
binary	Set to 1 if storing binary data, 0 if string

Returns

rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError, rmonError

rmonReadPersistentFLASH()

RTCU architecture: All
Called in: Connected State

Synopsis

rmonRet __stdcall rmonReadPersistentFLASH(HRMONCON hCon, int entry, char *data, int *length, int binary);

Description

Read FLASH based persistent entry
This function reads a specific entry from FLASH based persistent memory in the RTCU. When called, you must specify what type of data you are expecting to read, if the specified type of data is not present, the function returns rmonNoData, otherwise the data and length are returned.
Data read with this function can be stored from VPL with the functions SaveString() and SaveData().

Input

hCon	Handle to connection
entry	Entry number, from 1 to 192
binary	Set to 1 if expecting binary data, 0 if string expected

Output

data	Buffer for data (must be large enough!)
length	The number of bytes read from the entry

Returns

rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError, rmonNoData, rmonError

rmonWritePersistentFLASH()

RTCU architecture: All
Called in: Connected State

Synopsis

rmonRet __stdcall rmonWritePersistentFLASH(HRMONCON hCon, int entry, char *data, int length, int binary);

Description

Write to FLASH based persistent entry.

This function writes either binary data or a string to a specific entry in the FLASH based persistent memory in the RTCU. When called, you must specify what type of data you are storing.
Data stored with this function, can be read from VPL with the functions LoadString() and LoadData().

Input

hCon	Handle to connection
entry	Entry number, from 1 to 192
data	The data to store
length	Length of data
binary	Set to 1 if storing binary data, 0 if string

Returns

rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError, rmonError

rmonGetXFLASHSize()

RTCU architecture: All
Called in: Connected State

Synopsis
rmonRet __stdcall rmonGetXFLASHSize(HRMONCON hCon, int *size);

Description
Get the number of entries in extended FLASH.
This function is identical to the VPL function GetFlashXSize().

Input

hCon	Handle to connection

Output

size	Number of entries in extended flash.

Returns
rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

rmonReadPersistentXFLASH()

RTCU architecture: All
Called in: Connected State

Synopsis
rmonRet __stdcall rmonReadPersistentXFLASH(HRMONCON hCon, int entry, char *data, int *length, int binary);

Description
Read extended FLASH based persistent entry

This function reads a specific entry from extended FLASH based persistent memory in the RTCU. When called, you must specify what type of data you are expecting to read, if the specified type of data is not present, the function returns rmonNoData, otherwise the data and length are returned.
Data read with this function can be stored from VPL with the functions SaveStringX() and SaveDataX().

Input

hCon	Handle to connection
entry	Entry number, from 1 to Size (Determined with the function rmonGetXFLASHSize)
binary	Set to 1 if expecting binary data, 0 if string expected

Output

data	Buffer for data (must be large enough!)
length	The number of bytes read from the entry

Returns
rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError. rmonError, rmonNoData

rmonWritePersistentXFLASH()

RTCU architecture: All
Called in: Connected State

Synopsis

rmonRet __stdcall rmonWritePersistentXFLASH(HRMONCON hCon, int entry, char *data, int length, int binary);

Description

Write to extended FLASH based persistent entry.
This function writes either binary data or a string to a specific entry in the extended FLASH based persistent memory in the RTCU. When called, you must specify what type of data you are storing.
Data stored with this function, can be read from VPL with the functions LoadStringX() and LoadDataX().

Input

hCon	Handle to connection
entry	Entry number, from 1 to Size (Determined with the function rmonGetXFLASHSize)
data	The data to store
length	Length of data
binary	Set to 1 if storing binary data, 0 if string

Returns

rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError. rmonError

Datalogger

The built-in datalogger of the RTCU device can be manipulated with this set of functions. The log can be read, searched and cleared etc. For a more detailed description of the datalogger in the RTCU devices, please refer to the online help for the RTCU IDE.

The read and write pointer is NOT shared with the VPL application

rmonLogFirst()	**RTCU architecture:** All **Called in:** Connected State
Synopsis	rmonRet __stdcall rmonLogFirst(HRMONCON hCon)
Description	Moves the current read pointer to the first (oldest) entry in the datalogger
Input	**hCon** — Handle to connection
Returns	rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

rmonLogLast()	**RTCU architecture:** All **Called in:** Connected State
Synopsis	rmonRet __stdcall rmonLogLast(HRMONCON hCon)
Description	Moves the current read pointer to the last (newest) entry in the datalogger..
Input	**hCon** — Handle to connection
Returns	rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

rmonLogReadExt()

RTCU architecture: All
Called in: Connected State

Synopsis

rmonRet __stdcall rmonLogReadExt(HRMONCON hCon, int operation, int* values_per_rec, int* entries_in_buffer, char* buffer);

Description

This function reads up to 146 log entries from the datalogger each time it is called. When it is called, the current read pointer is either incremented or decremented, according to the operation parameter.

Input

hCon	Handle to connection
operation	RMONLOGGET_NEXT or RMONLOGGET_PREV

Output

values_per_rec	Number of values in each log entry, maximum 8
entries_in_buffer	Number of entries in the buffer, maximum 146
buffer	Buffer containing the read entries. The entries are placed directly after each other, and the format of each entry can be seen below. The required size of the buffer is depends on the type of connection: Cable (standard data size): 240 byte. Remote (standard data size): 480 byte. When large data is enabled(see **Error! Reference source not found.**): Cable (large data size): 1024 byte.

Returns

rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError, rmonNoData, rmonNoMoreData

Entry format:

Data type	Size (byte)	Description
signed char	1	Year relative to year 2000
unsigned char	1	Month, 1..12
unsigned char	1	Date, 1..31
unsigned char	1	Hour, 0..23
unsigned char	1	Minute, 0..59
unsigned char	1	Second, 0..59
unsigned char	1	Tag
int[n]	0-32	Log values, where n is the number of values in each entry (0-8)

rmonLogGetValuesPerRecord()

RTCU architecture: All
Called in: Connected State

Synopsis

rmonRet __stdcall rmonLogGetValuesPerRecord(HRMONCON hCon, int* numberofvalues)

Description

This function returns information about how many values (up to 8) are stored at each record in the datalogger (is configured via the VPL program in the RTCU device)

Input

hCon	Handle to connection

Output

numberofvalues	The number of values stored in each record in the datalogger.

Returns

rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

rmonLogClear()	RTCU architecture: All Called in: Connected State

Synopsis	rmonRet __stdcall rmonLogClear(HRMONCON hCon)	
Description	Clears data in the RTCU datalogger. The current datastructure in the RTCU datalogger is maintained.	
Input	**hCon**	Handle to connection
Returns	rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError, rmonNotInit	

rmonLogGotoLinsec()	RTCU architecture: All Called in: Connected State

Synopsis	rmonRet __stdcall rmonLogGotoLinsec(HRMONCON hCon, struct rmonRTCTime timestamp, unsigned char direction)	
Description	rmonLogGotoLinsec will search for an entry in the datalogger, that matches the specified timestamp, and if no match is found, it will select the nearest record (if any). It is possible to specify the search direction as either forward or backward.	
Input	**hCon**	Handle to connection
	timestamp	Complete time of record to search for, Please refer to the definition of rmonRTCTime below
	direction	False (0) means backwards search, True (different from 0) means forward search
Returns	rmonOK, rmonComError, rmonIllegalHandle, rmonNoData	

rmonLogReadByTag()

RTCU architecture: All
Called in: Connected State

Synopsis

rmonRet __stdcall rmonLogReadByTag(HRMONCON hCon, int operation, unsigned char tag, int* values_per_rec, int* entries_in_buffer, char* buffer);

Description

This function reads up to 146 log entries from the datalogger each time it is called. When it is called, the current read pointer is either incremented or decremented, according to the operation parameter.

Input

hCon	Handle to connection
operation	RMONLOGGET_NEXT or RMONLOGGET_PREV
tag	The tag that is used to filter datalog entries

Output

values_per_rec	Number of values in each log entry, maximum 8
entries_in_buffer	Number of entries in the buffer, maximum 146
buffer	Buffer containing the read entries. The entries are placed directly after each other, and the format of each entry can be seen below. The required size of the buffer is depends on the type of connection: Cable (standard data size): 240 byte. Remote (standard data size): 480 byte. When large data is enabled(see rmonEnableLargeData()): Cable (large data size): 1024 byte.

Returns

rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError, rmonNoData, rmonNoMoreData

Entry format:

Data type	Size (byte)	Description
signed char	1	Year relative to year 2000
unsigned char	1	Month, 1..12
unsigned char	1	Date, 1..31
unsigned char	1	Hour, 0..23
unsigned char	1	Minute, 0..59
unsigned char	1	Second, 0..59
unsigned char	1	Tag
int[n]	0-32	Log values, where n is the number of values in each entry (0-8)

rmonLogSeek()

RTCU architecture: All
Called in: Connected State

Synopsis

rmonRet __stdcall rmonLogSeek(HRMONCON hCon, short tag, short n)

Description

rmonLogSeek will search for an entry in the datalogger, that matches the specified tag, and move n records from there. The n parameter determines the direction of the search.

Input

hCon	Handle to connection
tag	The tag to search for.
n	The number of records to move. > 0 (zero): Seek forward. = 0 (zero): No effect. < 0 (zero): Seek Backwards.

Returns	rmonOK, rmonComError, rmonIllegalHandle, rmonNoData

I/O system functions

This group of functions allows access to the physical in- and outputs of the RTCU device, as well as the memory I/O system. The memory I/O system is accessible thru the VPL program as normal VAR_INPUT/VAR_OUTPUT variables. The variables must be configured in the RTCU IDE job configuration to either “To memory” or “From Memory”, depending on if they are declared as VAR_INPUT or VAR_OUTPUT variables.
The memory I/O system is 4096 elements.

rmonReadIOMemory()

RTCU architecture: All
Called in: Connected State

Synopsis

rmonRet __stdcall rmonReadIOMemory(HRMONCON hCon, int location, int count, int type, void *data);

Description

This function read from the memory I/O system in the RTCU. It is possible to indicate what type of data is stored at each location read; this is to help the function minimizing communication traffic.

Input

hCon	Handle to connection
location	This is the start location to read from, 0 based.
count	Number of memory locations to read
type	1=BOOL, 2=SINT, 3=INT, 4=DINT, this is the type of data to read from each location

Output

data	Data read from the RTCU will be stored in this buffer (must be ‘count’ number long, and of the same type at ‘type’ specifies (BOOL and SINT is 1 byte, INT is 2 bytes and DINT is 4 bytes) The required size of the buffer is depends on the type of connection: Cable (standard data size): 240 byte. Remote (standard data size): 480 byte. When large data is enabled(see **Error! Reference source not found.**): Cable (large data size): 1024 byte.

Returns

rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

rmonWriteIOMemory()

RTCU architecture: All
Called in: Connected State

Synopsis

rmonRet __stdcall rmonWriteIOMemory(HRMONCON hCon, int location, int count, int type, void *data);

Description

This function writes to the memory I/O system in the RTCU. It is possible to indicate what type of data is to be stored at each location; this is to help the function minimizing communication traffic. Please note that if the location(s) written to, is also used as VAR_OUTPUT variables by the VPL program in the RTCU device, the RTCU and this function will both write to the same location, in which case the writing done by this function, will be overwritten by the RTCU device itself (all I/O configured variables will be updated in each scan of the VPL program in the RTCU device).

Input

hCon	Handle to connection
location	This is the start location to write to, 0 based.
count	Number of memory locations to write
type	1=BOOL, 2=SINT, 3=INT, 4=DINT
data	Data written to the RTCU is taken from this buffer (must be 'count' number long, and of the same type at 'type' specifies (BOOL and SINT is 1 byte, INT is 2 bytes and DINT is 4 bytes) The required size of the buffer is depends on the type of connection: Cable (standard data size): 240 byte. Remote (standard data size): 480 byte. When large data is enabled(see rmonEnableLargeData()**Error! Reference source not found.**): Cable (large data size): 1024 byte.

Returns

rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError, rmonError

rmonGetIOState()

RTCU architecture: All
Called in: Connected State

Synopsis

rmonRet __stdcall rmonGetIOState(HRMONCON hCon, int iotype, int ioindex, int* value)

Description

This function is used to read the state of the physical in- and output signals in the RTCU device. The 'iotype' indicates which input/output you are reading from, and 'ioindex' indicates which input- or output number you are reading.

Input

hCon	Handle to connection
iotype	Select the type of I/O system you want to read from, see below
ioindex	Valid index of IO to get value from. Starts with index 0

Output

value	Input or output value

Returns

rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

Iotype:

Symbolic name	Value	Description
RMON_IOTYPE_DIN	1	Digital input
RMON_IOTYPE_DOUT	2	Digital output
RMON_IOTYPE_AIN	3	Analog input
RMON_IOTYPE_AOUT	4	Analog output
RMON_IOTYPE_LED	5	LED
RMON_IOTYPE_DIPSW	6	Dip switch

rmonSetIOState()

RTCU architecture: All
Called in: Connected State

Synopsis rmonRet __stdcall rmonSetIOState(HRMONCON hCon, int iotype, int ioindex, int value)

Description This function is used to set the status of the physical output signals in the RTCU device. The 'iotype' indicated which Output you are writing to, and 'ioindex' indicates which output number you are writing to.
Please note that if the output written to, is also used as a VAR_OUTPUT variable by the VPL program in the RTCU device, the RTCU and this function will both write to the same output, in which case the writing done by this function will be overwritten by the RTCU device itself (all I/O configured variables will be updated in each scan of the VPL program in the RTCU device).

Input

hCon	**Handle to connection**
Iotype	Select the type of output system you want to set, see below
Ioindex	This is the number of the output, 0 based.
value	Value to set

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

Iotype:

Symbolic name	Value	Description
RMON_IOTYPE_DOUT	2	Digital output
RMON_IOTYPE_AOUT	4	Analog output
RMON_IOTYPE_LED	5	LED

rmonGetIOCount()

RTCU architecture: X32, NX32
Called in: Connected State

Synopsis rmonRet __stdcall rmonGetIOCount(HRMONCON hCon, struct rmonIOCount *data)

Description This function is used to read the number of inputs and outputs on the device. (Onboard and external)

Input

hCon	Handle to connection

Output

data	A structure that contains the I/O count

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

```
typedef struct {
  unsigned char    NumberOfAI;
  unsigned char    NumberOfAO;
  unsigned char    NumberOfDI;
  unsigned char    NumberOfDO;
  unsigned char    NumberOfDIPSW;
  unsigned char    NumberOfLED;
  unsigned char    NumberOfExtAI;
  unsigned char    NumberOfExtAO;
  unsigned char    NumberOfExtDI;
  unsigned char    NumberOfExtDO;
  unsigned char    NumberOfExtDIPSW;
  unsigned char    NumberOfExtLED;
} rmonIOCount;
```

Real time clock

Functions that will read and set the realtime clock in the RTCU device.

The two functions, rmonGetRTC() and rmonSetRTC, both uses the following structure:

```
struct rmonRTCTime {
   unsigned short year;        // 2000..2048
   unsigned char  month;       // 01..12
   unsigned char  date;        // 01..31
   unsigned char  day;         // 01..07
   unsigned char  hour;        // 00..23
   unsigned char  minute;      // 00..59
   unsigned char  second;      // 00..59
};
```

rmonGetRTC()	**RTCU architecture:** All **Called in:** Connected State
Synopsis	rmonRet __stdcall rmonGetRTC(HRMONCON hCon, struct rmonRTCTime *rtc)
Description	Reads the real time clock on the RTCU device. Returns the current time in a rmonRTCTime structure.
Input	**hCon** — Handle to connection
Output	**rtc** — Please refer to the definition of rmonRTCTime above
Returns	rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

rmonSetRTC()	**RTCU architecture:** All **Called in:** Connected State
Synopsis	rmonRet _stdcall rmonSetRTC(HRMONCON hCon, struct rmonRTCTime *rtc)
Description	Sets the real time clock on the RTCU device. The time is supplied in a rmonRTCTime structure.
Input	**hCon** — Handle to connection **rtc** — Please refer to the definition of rmonRTCTime above
Returns	rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

GSM/SMS functions

This is a set of functions, which allows you to read and set various parameters used in the RTCU devices interaction with the GSM module.

Also the functions allow you to send and receive “fake” SMS messages to/from the RTCU device. If the VPL program in the device sends an sms message to phone number “9999”, using the VPL function gsmSendSMS() / gsmSendPDU(), the message will be received by the library, and the message will then available thru the function rmonReceiveSMS(). The same goes for your application, it can call rmonSendSMS(), and the VPL program in the RTCU can use gsmIncomingSMS() / gsmIncomingPDU() to receive this message sent from your application. This is a very easy to use way of communicating small messages back and forth between your PC application and the VPL application of the RTCU device.

rmonGetIMEI()

RTCU architecture: All
Called in: Connected State

Synopsis rmonRet __stdcall rmonGetIMEI(HRMONCON hCon, char *IMEInumber, int bufsize)

Description is used for fetching the IMEI number of the GSM module

Input

hCon	Handle to connection
bufsize	Number of characters to read. If bufsize exceeds the number of characters in the number, only the number of characters present will be put in the output buffer.

Output

IMEInumber	This is where the information will be stored

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

rmonGetIMSI()

RTCU architecture: All
Called in: Connected State

Synopsis rmonRet __stdcall rmonGetIMSI(HRMONCON hCon, char *IMSInumber, int bufsize)

Description is used for fetching the IMSI number of the GSM module

Input

hCon	Handle to connection
bufsize	Number of characters to read. If bufsize exceeds the number of characters in the number, only the number of characters present will be put in the output buffer.

Output

IMSInumber	This is where the information will be stored

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

rmonGetICCID()

RTCU architecture: All
Called in: Connected State

Synopsis rmonRet __stdcall rmonGetICCID(HRMONCON hCon, char *ICCIDnumber, int bufsize)

Description () is used for fetching the ICCID number of the GSM module

Input

hCon	Handle to connection
bufsize	Number of characters to read. If bufsize exceeds the number of characters in the number, only the number of characters present will be put in the output buffer.

Output

ICCIDnumber	This is where the information will be stored

Returns	rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

rmonSendSMS()

RTCU architecture: All
Called in: Connected State

Synopsis

rmonRet __stdcall rmonSendSMS(HRMONCON hCon, int smstype, int messageLength, const char* message)

Description

The PC application can send “fake” SMS messages to the RTCU device using this function. The RTCU will receive SMS messages with the gsmIncomingSMS() / gsmIncomingPDU(), and when the message is sent to the RTCU using this function, the .phonenumber variable in the gsmIncomingSMS() / gsmIncomingPDU() will indicate “9999” as the originator of the message.

Input

hCon	Handle to connection
smstype	Type of SMS message received, see below
messageLength	Only used when smstype is RMONSMS_BINARY
message	Zero terminated AZCII string when smstype is RMONSMS_TEXT. If smstype is RMONSMS_BINARY messageLength specifies length of data in message.

Returns

rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

smstype:

Symbolic name	Value	Description
RMONSMS_TEXT	0	Text based SMS message
RMONSMS_BINARY	1	Binary SMS message

rmonReceiveSMS()

RTCU architecture: All
Called in: Connected State

Synopsis

rmonRet __stdcall rmonReceiveSMS(HRMONCON hCon, int* smstype, int* dataLength, char* data)

Description

When the connected RTCU device sends a SMS message to phonenumber “9999” using the either gsmSendSMS() or gsmSendPDU(), this function will receive these messages. Please note that this function is blocking, it will first return when a message is received.

Input

gCon	Handle to connection

Output

smstype	Type of SMS message received, see below
dataLength	Only used when smstype = RMONSMS_BINARY
data	Zero terminated ASCII string when smstype = RMONSMS_TEXT

Returns

rmonOK, rmonIllegalHandle, rmonTargetError, rmonNoData

smstype:

Symbolic name	Value	Description
RMONSMS_TEXT	0	Text based SMS message
RMONSMS_BINARY	1	Binary SMS message

rmonReceiveSMSEnable()

RTCU architecture: All
Called in: Connected State

Synopsis rmonRet __stdcall rmonReceiveSMSEnable(HRMONCON hCon, int enable)

Description Using this function, it is possible to either enable or disable reception of the “fake” SMS messages from the RTCU device by the function.

Note, if disabling, the rmonReceiveSMS() will still block, waiting for a message to arrive.

Input

hCon	Handle to connection
enable	0=disable, 1=enable

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

rmonGetGSMSignalLevel()

RTCU architecture: All
Called in: Connected State

Synopsis rmonRet __stdcall rmonGetGSMSignalLevel(HRMONCON hCon, int* signal)

Description This functions returns the GSM signal level. (This function does the same as the VPL function gsmSignalLevel()).

Input

hCon	Handle to connection

Output

signal	The GSM signal strength or 0 (zero) if not connected.

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

rmonSetAllowedCallerList()

RTCU architecture: All
Called in: Connected State

Synopsis rmonRet __stdcall rmonSetAllowedCallerList (HRMONCON hCon, const char numbers[81])

Description Sets list of allowed phone numbers that can make incoming data calls to the RTCU. Phone numbers must be separated with the ”,” character.
The list of allowed callers can also be set from the RTCU IDE (menu: Device -> Configuration -> GSM options).
(This function does the same as the VPL function gsmSetListOfCallers()).

Input

hCon	Handle to connection
numbers	List of phonenumbers separated by “,” character

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

rmonGetAllowedCallerList()

RTCU architecture: All
Called in: Connected State

Synopsis
rmonRet __stdcall rmonGetAllowedCallerList(HRMONCON hCon, char numbers[81])

Description
Fetches list of allowed caller numbers set in rmonSetAllowedCallerList().
The list of allowed callers my also be fetched from the RTCU IDE (menu: Device -> Configuration -> GSM options).
(This function does the same as the VPL function gsmGetListOfCallers()).

Input

hCon	Handle to connection

Output

numbers	List of allowed caller numbers separated by "," character.

Returns
rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

rmonSetGSMPIN()

RTCU architecture: All
Called in: Connected State

Synopsis
rmonRet __stdcall rmonSetGSMPIN (HRMONCON hCon, const char pin[5])

Description
Sets the SIM PIN code to use for the SIM card in the RTCU device.
This does NOT change the PIN code on the SIM card, it simply tells the RTCU device which PIN code to use when powering up the GSM module !

An empty string will disable use of PIN code (SIM PIN code must be disabled on the SIM card, use a normal mobile telephone for doing this)
Specifying a wrong GSM pin code will cause a RTCU fault.

If the RTCU is restarted more than 3 times with the wrong SIM pin code, the SIM card will be locked, and it must be unlocked in a normal mobile phone, using the GSM operator supplied PUK code !

Please notice the SIM PIN code may also be set using the RTCU IDE (menu: Device -> Configuration -> GSM options)
(This function does the same as the VPL function gsmSetPin()).

Input

hCon	Handle to connection
pin	New SIM PIN code to be set

Returns
rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

rmonGetGSMPIN()

RTCU architecture: All
Called in: Connected State

Synopsis
rmonRet __stdcall rmonGetGSMPIN (HRMONCON hCon, const char pin[5])

Description
Fetches the GSM SIM PIN code from the RTCU (see rmonSetGSMPin() above).
An empty string denotes that the PIN code has been disabled.

Input

hCon	Handle to connection

Output

pin	Current SIM PIN code. Empty string specifies the PIN code to be currently disabled

Returns
rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

Filesystem functions

This is a set of functions that offers a broad range of operations on the file system present in the RTCU device.
The filesystem error-codes start from 100, but are otherwise identical to the VPL ones:

Symbolic name	Value	Description
RMONFS_INVALIDDRIVE	101	The media is not opened
RMONFS_NOTFOUND	105	The directory or file is not found
RMONFS_DUPLICATED	106	The directory or file already exist
RMONFS_NOMOREENTRY	107	The media is full
RMONFS_NOTOPEN	108	The file is not open
RMONFS_LOCKED	112	The file is in use
RMONFS_NOTEMPTY	114	The directory is not empty
RMONFS_CARDREMOVED	116	The media is not present
RMONFS_ONDRIVE	117	Media communication error
RMONFS_BUSY	122	The media is busy
RMONFS_WRITEPROTECT	123	The media is write-protected
RMONFS_FILEACCESS	138	The file is no longer accessible and must be closed
RMONFS_EXTENSION	139	The media is busy with Platform extension

The filesystem functions has these limitations in addition to the ones for the file system in general (See the RTCU IDE Online-help/Manual):

- A client can only have one open file at any given time. If more files are opened, the already open file is closed.
- The working directory is shared with all other clients connected to the device. Because of this it is recommended to use absolute paths where possible.

The media available:

Media ID	Drive	Description
0	A:	The SD-CARD.
1	B:	The Internal drive.
2	P:	The Intellisync Project drive.
3	U:	The USB Mass storage drive.*

* The USB Mass storage drive is only available on NX32L devices with an USB host port.

rmonMediaPresent()

RTCU architecture: All
Called in: Connected State

Synopsis rmonRet __stdcall rmonMediaPresent(HRMONCON hCon, int media, int* state, int* fserr)

Description Queries whether the Media is present or not.

Input

hCon	Handle to connection
media	The media ID. (See introduction for available media)

Output

State	=0 (zero) if media is not present. <>0 (zero) if media is present
Fserr	Error code from the filesystem

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

rmonMediaWriteprotected()

RTCU architecture: All
Called in: Connected State

Synopsis rmonRet __stdcall rmonMediaWriteprotected (HRMONCON hCon, int media, int* state, int* fserr)

Description Queries whether the Media is write protected or not.

Input

hCon	Handle to connection
media	The media ID. (See introduction for available media)

Output

State	=0 (zero) if media is not write protected. <>0 (zero) if media is write protected.
Fserr	Error code from the filesystem

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

rmonMediaOpen()

RTCU architecture: All
Called in: Connected State

Synopsis rmonRet __stdcall rmonMediaopen (HRMONCON hCon, int media, int* fserr)

Description Open the media for use with the filesystem.

Input

hCon	Handle to connection
media	The media ID. (See introduction for available media)

Output

Fserr	Error code from the filesystem

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

rmonMediaClose()

RTCU architecture: All
Called in: Connected State

Synopsis rmonRet __stdcall rmonMediaopen (HRMONCON hCon, int media, int* fserr)

Description Close the media for use with the filesystem.

Input

hCon	Handle to connection
media	The media ID. (See introduction for available media)

Output

Fserr	Error code from the filesystem

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

rmonMediaQuickformat()

RTCU architecture: All
Called in: Connected State

Synopsis rmonRet __stdcall rmonMediaQuickformat (HRMONCON hCon, int media, int* fserr)

Description Quick formats the media.

Input

hCon	Handle to connection
media	The media ID. (See introduction for available media)

Output

Fserr	Error code from the filesystem.

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

rmonMediaQuickformatX()

RTCU architecture: All
Called in: Connected State

Synopsis rmonRet __stdcall rmonMediaQuickformatX (HRMONCON hCon, int media, int flags, int* fserr)

Description Quick formats the media.

Input

hCon	Handle to connection
media	The media ID. (See introduction for available media)
flags	Option flags to control the behaviour of the format. (Only used by NX32L devices)

Output

Fserr	Error code from the filesystem.

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

flags:

Symbolic name	Value	Description
RMONFS_FMTFLAG_FULL	1	Perform full format of drive

rmonMediaEject()

RTCU architecture: All
Called in: Connected State

Synopsis rmonRet __stdcall rmonMediaEject(HRMONCON hCon, int media, int* fserr)

Description Eject the media.

Input

hCon	Handle to connection
media	The media ID. (See introduction for available media)

Output

Fserr	Error code from the filesystem.

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

rmonMediaInformation()

RTCU architecture: All
Called in: Connected State

Synopsis rmonRet __stdcall rmonMediaInformation(HRMONCON hCon, struct rmonMediaInfo media[8])

Description Queries the device for which media is mounted, what type of media it is, and the capacity of the media.
The media type can be one of the following:

0	Not mounted
1	SD-CARD
2	Internal FLASH
3	Intellisync Project Drive
4	USB Mass storage drive

Input

hCon	Handle to connection

Output

media	An array of media information structures.

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

```
struct rmonMediaInfo {
  unsigned char       Type;   // Type of media
  unsigned long       Size;   // Size of the media, lower 32bits
  unsigned long       SizeHi; // Size of the media, upper 32bits
};
```

rmonMediaSize()

RTCU architecture: All
Called in: Connected State

Synopsis rmonRet __stdcall rmonMediaSize(HRMONCON hCon, int media, unsigned long* SizeLo, unsigned long* SizeHi, unsigned long* FreeLo, unsigned long* FreeHi)

Description Retrieve the total size and free size of a media.

Input

hCon	Handle to connection
media	The media ID. (See introduction for available media)

Output

SizeLo	The total size of the media, lower 32 bits.
SizeHi	The total size of the media, upper 32 bits.
FreeLo	The free size of the media, lower 32 bits.
FreeHi	The free size of the media, upper 32 bits.

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

rmonFSStatusLED()

RTCU architecture: All
Called in: Connected State

Synopsis rmonRet __stdcall rmonFSStatusLED(HRMONCON hCon, int enable, int* fserr)

Description Using this function, it is possible to either enable or disable the filesystem status LED's.

Input

hCon	Handle to connection
Enable	0=disable, 1=enable

Output

Fserr	Error code from the filesystem

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

rmonDirCreate()

RTCU architecture: All
Called in: Connected State

Synopsis rmonRet __stdcall rmonDirCreate(HRMONCON hCon, const char name[61], int* fserr)

Description Create a new directory.

Input

hCon	Handle to connection
Name	Name of the directory to create. (60 characters + 0 (zero) terminator) Both absolute and relative path can be used.

Output

Fserr	Error code from the filesystem.

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

rmonDirChange()

RTCU architecture: All
Called in: Connected State

Synopsis rmonRet __stdcall rmonDirChange(HRMONCON hCon, const char path[61], int* fserr)

Description Change the working directory.

Input

hCon	Handle to connection
Path	Path to the new working directory. (60 characters + 0 (zero) terminator) Both absolute and relative path can be used.

Output

Fserr	Error code from the filesystem.

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

rmonDirCurrent()

RTCU architecture: All
Called in: Connected State

Synopsis rmonRet __stdcall rmonDirCurrent(HRMONCON hCon, char path[61], int* fserr)

Description Retrieve the absolute path to the working directory

Input

hCon	Handle to connection

Output

Path	The absolute path to the working directory. (60 characters + 0 (zero) terminator)
Fserr	Error code from the file system.

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

rmonDirCatalog()

RTCU architecture: All
Called in: Connected State

Synopsis
rmonRet __stdcall rmonDirCatalog(HRMONCON hCon, short index, char name[15], struct rmonRTCTime* time, long* length, int* fserr)

Description
Retrieves the information of an entry in the working directory.

Input

hCon	Handle to connection
Index	The index of the directory entry to retrieve.

Output

Name	The name of the directory entry. (14 characters + zero terminator)
Time	The creation time of the file. Uses the same structure as rmonGetRTC. Not used for directories.
Length	The size of the file. Not used for directories.
Fserr	Error code from the filesystem.

Returns
rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

rmonDirCatalogX()

RTCU architecture: All
Called in: Connected State

Synopsis
rmonRet __stdcall rmonDirCatalog(HRMONCON hCon, char* wild, rmoncbdirentry pfunc, void* uptr, int* fserr)

Description
Retrieves the information of all the entries in the working directory that matches the provided wildcard string

Input

hCon	Handle to connection
wild	Wildcard string to search for. The wildcards ? for any one character and * for any number of characters are supported.
pfunc	Pointer to callback function which is called for each directory item.
uptr	Pointer to user argument used in callback function

Output

fserr	Error code from the filesystem.

Returns
rmonOK, rmonComError, rmonIllegalHandle, rmonError, rmonCancelled

The call-back function is defined as follows:
typedef int (__stdcall *rmoncbdirentry)(void* uptr, const char* filename, unsigned char type, long timestamp, unsigned long size);

Type:

Symbolic name	Value	Description
RMONFS_TYPE_FILE	0	File
RMONFS_TYPE_FOLDER	1	Directory

rmonDirDelete()	**RTCU architecture:** All **Called in:** Connected State

Synopsis rmonRet __stdcall rmonDirDelete(HRMONCON hCon, const char name[61], int* fserr)

Description Delete a directory.

Input

hCon	Handle to connection
Name	The name of the directory. (60 characters + zero terminator) Both absolute and relative path can be used.

Output

Fserr	Error code from the filesystem.

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

rmonFileCreate()	**RTCU architecture:** All **Called in:** Connected State

Synopsis rmonRet __stdcall rmonFileCreate(HRMONCON hCon, const char name[61], int* fserr)

Description Creates a new file. If a file is already open, it will be closed before the new file is created.

Input

hCon	Handle to connection
Name	The name of the file to create. (60 characters + zero terminator) Both absolute and relative path can be used.

Output

Fserr	Error code from the filesystem.

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

rmonFileOpen()	**RTCU architecture:** All **Called in:** Connected State

Synopsis rmonRet __stdcall rmonFileOpen(HRMONCON hCon, const char name[61], int* fserr)

Description Opens a file. If a file is already open, it will be closed before the new file is opened.

Input

HCon	Handle to connection
Name	The name of the file to create. (60 characters + zero terminator) Both absolute and relative path can be used.

Output

Fserr	Error code from the filesystem.

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

rmonFileExists()

RTCU architecture: All
Called in: Connected State

Synopsis rmonRet __stdcall rmonFileExists(HRMONCON hCon, const char name[61], char* state, int* fserr)

Description Query whether a file exists.

Input

hCon	Handle to connection
Name	The name of the file to create. (60 characters + zero terminator) Both absolute and relative path can be used.

Output

State	=0 (zero) if file does not exist. <>0 (zero) if file does exist.
Fserr	Error code from the filesystem.

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

rmonFileRename()

RTCU architecture: All
Called in: Connected State

Synopsis rmonRet __stdcall rmonFileRename(HRMONCON hCon, const char name_old[61], const char name_new[13], int* fserr)

Description Renames a file

Input

hCon	Handle to connection
Name_new	The new name of the file. (12 characters + zero terminator)
Name_old	The name of the file to rename. Both absolute and relative paths can be used (60 characters + zero terminator)

Output

Fserr	Error code from the filesystem.

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

rmonFileDelete()

RTCU architecture: All
Called in: Connected State

Synopsis rmonRet __stdcall rmonFileDelete(HRMONCON hCon, const char name[61], int* fserr)

Description Delete a file.

Input

hCon	Handle to connection
Name	The name of the file to create. (60 characters + zero terminator) Both absolute and relative path can be used.

Output

State	=0 (zero) if file does not exist. <>0 (zero) if file does exist.
Fserr	Error code from the filesystem.

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

rmonFileStatus()

RTCU architecture: All
Called in: Connected State

Synopsis rmonRet __stdcall rmonFileStatus(HRMONCON hCon, int* status, int* fserr)

Description Retrieve the status of the open file.

Input

hCon	Handle to connection

Output

Status	The status of the file. Identical to the return value of the fsFileStatus VPL function.
Fserr	Error code from the filesystem.

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

rmonFileGetInfo()

RTCU architecture: All
Called in: Connected State

Synopsis rmonRet __stdcall rmonFileGetInfo(HRMONCON hCon, const char name[61], struct rmonRTCTime* time, long* length, int* fserr)

Description Retrieve the size and creation timestamp of a file.

Input

hCon	Handle to connection
name	The name of the file. (60 characters + zero terminator) Both absolute and relative path can be used.

Output

Time	The creation timestamp. Please refer to the definition of rmonRTCTime above (Page 40)
Length	The size of the file in bytes.
Fserr	Error code from the filesystem.

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

rmonFileSeek()

RTCU architecture: All
Called in: Connected State

Synopsis rmonRet __stdcall rmonFileSeek(HRMONCON hCon, long offset, int* fserr)

Description Moves the file pointer.

Input

hCon	Handle to connection
Offset	The new position relative to the Start of file. >0 (zero) – Position in file. =0 (zero) – Start of file. -1 - End of file.

Output

Fserr	Error code from the filesystem.

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

rmonFilePosition()

RTCU architecture: All
Called in: Connected State

Synopsis rmonRet __stdcall rmonFilePosition(HRMONCON hCon, long* position, int* fserr)

Description Retrieve the file pointer position of the open file.

Input

hCon	Handle to connection

Output

Position	The file pointer position
Fserr	Error code from the filesystem.

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

rmonFileRead()

RTCU architecture: All
Called in: Connected State

Synopsis rmonRet __stdcall rmonFileRead(HRMONCON hCon, int elemcnt, char* buffer, int* elemread, int* fserr, rmoncbprogress pfunc, void* uptr)

Description Read a block of data from file.

Input

hCon	Handle to connection
Elemcnt	The number of bytes to read from file.
Pfunc	Pointer to function where progress is reported.
Arg	Pointer to user argument used when reporting progress.

Output

Buffer	The buffer where the data read from the file is stored.
elemread	The number of bytes read from file.
Fserr	Error code from the filesystem.

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

rmonFileReadString()

RTCU architecture: All
Called in: Connected State

Synopsis rmonRet __stdcall rmonFileReadString(HRMONCON hCon, char str[241], int* elemread, int* fserr)

Description Reads a string from the file. The function will read until a <CR><LF> termination sequence is found or the buffer is full (240 characters).

Input

hCon	Handle to connection

Output

str	The buffer where the string read from the file is stored. (240 characters + zero terminator)
elemread	The number of bytes read from file.
Fserr	Error code from the filesystem.

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

rmonFileWrite()

RTCU architecture: All
Called in: Connected State

Synopsis rmonRet __stdcall rmonFileWrite(HRMONCON hCon, int elemcnt, char* buffer, int* elemwr, int* fserr, rmoncbprogress pfunc, void* uptr)

Description Write a block of data to file.

Input

hCon	Handle to connection
elemcnt	The number of bytes to write to file.
Buffer	The buffer where the data to write is stored.
pfunc	Pointer to function where progress is reported.
uptr	Pointer to user argument used when reporting progress.

Output

elemwr	The number of bytes written to file
Fserr	Error code from the filesystem.

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

rmonFileWriteString()

RTCU architecture: All
Called in: Connected State

Synopsis rmonRet __stdcall rmonFileWriteString(HRMONCON hCon, const char str[241], int* elemwr, int* fserr)

Description Write a string to file.

Input

hCon	Handle to connection
str	The string to write to file. (240 characters + zero terminator)

Output

elemwr	The number of bytes written to file
Fserr	Error code from the filesystem.

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

rmonFileWriteStringNL()

RTCU architecture: All
Called in: Connected State

Synopsis rmonRet __stdcall rmonFileWriteStringNL(HRMONCON hCon, const char str[241], int* elemwr, int* fserr)

Description Write a string to file. <CR><LF> are appended.

Input

hCon	Handle to connection
str	The string to write to file. (240 characters + zero terminator)

Output

elemwr	The number of bytes written to file
Fserr	Error code from the filesystem.

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

rmonFileClose()	RTCU architecture: All Called in: Connected State

Synopsis	rmonRet __stdcall rmonFileClose(HRMONCON hCon, int* fserr)	
Description	Close the file.	
Input	**hCon**	Handle to connection
Output	**Fserr**	Error code from the filesystem.
Returns	rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError	

rmonFileFlush()	RTCU architecture: All Called in: Connected State

Synopsis	rmonRet __stdcall rmonFileFlush(HRMONCON hCon, int* fserr)	
Description	Flush cached write operations to media.	
Input	**hCon**	Handle to connection
Output	**Fserr**	Error code from the filesystem.
Returns	rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError	

Security functions

The certificates of the RTCU device can be manipulated by this set of functions.
For a more detailed description of security in the RTCU devices, please refer to the online help for the RTCU IDE.

rmonSecurityImport()

RTCU architecture: NX32L
Called in: Connected State

Synopsis rmonRet __stdcall rmonSecurityImport(HRMONCON hCon, const char* name, const char* filename_cert, const char* filename_key, const int replace)

Description Import a certificate to the.device.

Input

hCon	Handle to connection
name	The name of the certificate
filename_cert	The file name of the certificate to import.
filename_key	Optional file name of private encryption key to include with the certificate.
replace	0 = Fail if a certificate with the name already exists, 1 = Replace existing certificate.

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonFileNotFound, rmonIllegalFile, rmonError

rmonSecurityRemove()

RTCU architecture: NX32L
Called in: Connected State

Synopsis rmonRet __stdcall rmonSecurityRemove(HRMONCON hCon, const char* name)

Description Remove a certificate from the device.

Input

hCon	Handle to connection
name	The name of the certificate

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonError

rmonSecurityInfo()

RTCU architecture: NX32L
Called in: Connected State

Synopsis rmonRet __stdcall rmonSecurityInfo(HRMONCON hCon, const char* name, rmoncbcertificate pfunc, void* arg)

Description Fetches information about a certificate from the.device.

Input

hCon	Handle to connection
name	The name of the certificate
pfunc	Pointer to callback function which is called with the certificate information
arg	Pointer to user argument used in callback function

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonError

The call-back function is defined as follows:

```
typedef int (__stdcall *rmoncbcertificate)(void* uptr, const char* name, unsigned char type,
unsigned key, const char* subject, const char* issuer, long linsec _from, long linsec_to);
```

System Object Storage functions

The System Object Storage (SOS) of the RTCU device can be manipulated by this set of functions. For a more detailed description of the SOS in the RTCU devices, please refer to the online help for the RTCU IDE.

Object table:

Symbolic name	Value	Description
RMONOBJ_TABLE_SYSTEM	1	The system object table
RMONOBJ_TABLE_USER	2	The user object table

Object types:

Symbolic name	Value	Description
RMONOBJ_TYPE_BOOL	1	Boolean value
RMONOBJ_TYPE_INT	2	Integer value
RMONOBJ_TYPE_STRING	3	String value
RMONOBJ_TYPE_DATA	4	Binary data value
RMONOBJ_TYPE_FLOAT	5	Floating point value
RMONOBJ_TYPE_DOUBLE	6	Double floating point value

rmonObjectRead()

RTCU architecture: NX32L
Called in: Connected State

Synopsis rmonRet __stdcall rmonObjectRead(HRMONCON hCon, const int table, const char* name, rmoncdobject output, void* uptr)

Description Fetches objects from the.device.

Input

hCon	Handle to connection
table	The object table to use
name	The name of the object. Wildcards can be used.
output	Pointer to callback function which is called for each object read
uptr	Pointer to user argument used in callback function

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonError, rmonNoData

The call-back function is defined as follows:
typedef int (__stdcall *rmoncbobject)(void* uptr, const char* name, int type, int size, unsigned char* data, unsigned long flags, int min_val, int max_val, const char* description);

uptr	User pointer
name	The name of the object
type	The type of the object. (See object types above)
size	The size of the data in bytes
data	The object data.
flags	Object flags. See table below.
min_val	The minimum value or the minimum length, of the object.
max_val	The maximum value or the maximum length, of the object.
description	A short description of the object.

0x0001	Read only object
0x0002	Encrypted object

rmonObjectReadX()

RTCU architecture: All
Called in: Connected State

Synopsis

rmonRet __stdcall rmonObjectRead(HRMONCON hCon, const int table, const char* name, rmoncdobjectx output, void* uptr)

Description

Fetches objects from the.device.

Input

hCon	Handle to connection
table	The object table to use
name	The name of the object. Wildcards can be used.
output	Pointer to callback function which is called for each object read
uptr	Pointer to user argument used in callback function

Returns

rmonOK, rmonComError, rmonIllegalHandle, rmonError, rmonNoData

The call-back function is defined as follows:
typedef int (__stdcall *rmoncbobjectx)(void* uptr, const char* name, int type, int size, unsigned char* data, unsigned long flags, int min_size, unsigned char* min_val, int max_size, unsigned char* max_val, const char* description);

uptr	User pointer
name	The name of the object
type	The type of the object. (See object types above)
size	The size of the data in bytes
data	The object data.
flags	Object flags. See table below.
min_size	The size of the min_val parameter in bytes.
min_val	Pointer to a buffer, containing the minimum value or the minimum length, of the object.
max_size	The size of the max_val parameter in bytes.
max_val	Pointer to a buffer, containing the maximum value or the maximum length, of the object.
description	A short description of the object.

0x0001	Read only object
0x0002	Encrypted object

rmonObjectWrite()

RTCU architecture: NX32L
Called in: Connected State

Synopsis

rmonRet __stdcall rmonObjectWrite(HRMONCON hCon, const int table, const short flags, const int count, rmonObjectInfo* data, int* index)

Description

Write objects to the.device.

Input

hCon	**Handle to connection**
table	The object table to use
flags	Flags to control the write operation. See below.
count	The number of objects in the array
data	Pointer to an array of objects to write

Output

index	**The index of the object which failed. 0 if error is not from an objects or no error**

Returns

rmonOK, rmonComError, rmonIllegalHandle, rmonError, rmonNoData

```
typedef struct {
  char             name[255];
  int              type;
  int              size;
  unsigned char*   data;
  unsigned long    flags;
  int              min_val;
  int              max_val;
  char             desc[81];
} rmonObjectInfo;
```

Object flags:

0x0001	Read only object
0x0002	Encrypted object

Write flags:

Symbolic name	Value	Description
RMONOBJ_FLAG_OVERWRITE	0x0001	Overwrite any existing objects
RMONOBJ_FLAG_IGNORE	0x0002	Only update existing objects.

rmonObjectWriteX()

RTCU architecture: NX32L
Called in: Connected State

Synopsis rmonRet __stdcall rmonObjectWriteX(HRMONCON hCon, const int table, const short flags, const int count, rmonObjectInfoX* data, int* index)

Description Write objects to the.device.

Input

hCon	Handle to connection
table	The object table to use
flags	Flags to control the write operation. See below.
count	The number of objects in the array
data	Pointer to an array of objects to write

Output

index	The index of the object which failed. 0 if error is not from an objects or no error

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonError, rmonNoData

```
typedef struct {
  char              name[255];
  int               type;
  int               size;
  unsigned char*    data;
  unsigned long     flags;
  int               min_size;
  unsigned char*    min_val;
  int               max_size;
  unsigned char*    max_val;
  char              desc[81];
} rmonObjectInfoX;
```

Object flags:

0x0001	Read only object
0x0002	Encrypted object

Write flags:

Symbolic name	Value	Description
RMONOBJ_FLAG_OVERWRITE	0x0001	Overwrite any existing objects
RMONOBJ_FLAG_IGNORE	0x0002	Only update existing objects.

rmonObjectErase()

RTCU architecture: NX32L
Called in: Connected State

Synopsis rmonRet __stdcall rmonObjectErase(HRMONCON hCon, const int table, const char* name)

Description Erase an object from the device.

Input

hCon	Handle to connection
table	The object table to use
name	The name of the object. Wildcards can be used.

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonError

Misc. functions

Below is a list of different “housekeeping” functions.

rmonReset()	**RTCU architecture:** All **Called in:** Connected State

Synopsis	rmonRet __stdcall rmonReset(HRMONCON hCon)
Description	This function will reset the connected RTCU device. If the RTCU device is remotely connected (RCH) the reset will be delayed until the connection is lost (by calling rmonDisconnect() etc). However, if the connection is thru a direct cable connection, the RTCU device executes the reset command immediately. The reset has the same effect as cycling power to the RTCU device, the VPL program starts executing from the start again. This can also be carried out from the RTCU IDE (menu: Device -> Execution -> Reset) (This function does the same as the VPL function boardReset()).
Input	**hCon** — Handle to connection
Returns	rmonOK, rmonComError, rmonIllegalHandle

rmonHalt()	**RTCU architecture:** All **Called in:** Connected State

Synopsis	rmonRet __stdcall rmonHalt(HRMONCON hCon)
Description	Stops the currently executing VPL program in the RTCU. This can also be carried out from the RTCU IDE (menu: Device -> Execution -> Halt) The RTCU device can be started again with the reset command (see above) or by cycling power.
Input	**hCon** — Handle to connection
Returns	rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

rmonGetSerialNumber()	**RTCU architecture:** All **Called in:** Connected State

Synopsis	rmonRet __stdcall rmonGetSerialNumber(HRMONCON hCon, unsigned long *SerialNumber)
Description	Returns the serial number of the connected RTCU.
Input	**hCon** — Handle to connection
Output	**SerialNumber** — The serialnumber of the RTCU device.
Returns	rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError, rmonIllegalTarget

rmonVer()	**RTCU architecture:** All **Called in:** Connected State

Synopsis rmonRet __stdcall rmonVer(HRMONCON hCon, int *ver)

Description Returns the Firmware version of the connected RTCU.
On NX32L devices, the major and minor part of the runtime version is returned. (see rmonGetDeviceInfo())
Note that this function must be used to determine if RTCU device is in monitor mode.

Input

hCon	Handle to connection

Output

ver	Firmware version, always different from 0 and scaled by 100 (Version 4.66 is returned as 466). Note that a version higher than 90.00 means that the RTCU is in monitor mode.

Returns rmonOK, rmonComError, rmonIllegalHandle

rmonSetPassword()	**RTCU architecture:** All **Called in:** Connected State

Synopsis rmonRet __stdcall rmonSetPassword (HRMONCON hCon, const char password[21])

Description Sets new password for access to RTCU.
This is the password that is to be used in rmonAuthenticate().
An empty string will disable password protection.
This can also be set from the RTCU IDE (menu: Device -> Configuration -> Set Password)

Input

hCon	Handle to connection
password	New password to be set (zero terminated ASCII String).

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

rmonGetTargetInfo()

RTCU architecture: All
Called in: Connected State

Synopsis

rmonRet __stdcall rmonGetTargetInfo(HRMONCON hCon, int* targetID, int* firmwareVer);

Description

Fetches RTCU type and firmware version from the RTCU.
On NX32L devices, the major and minor part of the runtime version is returned. (see rmonGetDeviceInfo())

Input

hCon	**Handle to connection**

Output

targetID	Please see the table below for a list of possible target ID's.
firmwareVer	Firmware version always different from 0 and scaled by 100 (Version 4.66 is returned as 466)

Returns

rmonOK, rmonComError, rmonIllegalHandle

Symbolic name	Value	Description
RMONTGT_ICP002	1	RTCU SA
RMONTGT_ICP003	2	RTCU DIN
RMONTGT_ICP004	4	RTCU D4
RMONTGT_ICP005	5	RTCU A5 / RTCU A5i
RMONTGT_ICP006	6	RTCU A6
RMONTGT_ICP007	7	RTCU M7
RMONTGT_ICP009	9	RTCU A9i
RMONTGT_ICP010	10	RTCU M10 Series
RMONTGT_ICP011	11	RTCU M11 Series
RMONTGT_ICP102	102	RTCU MX2 Series
RMONTGT_ICP103	103	RTCU DX4i mk2
RMONTGT_ICP104	104	RTCU DX4 Series
RMONTGT_ICP105	105	RTCU CX1 Series
RMONTGT_ICP106	106	RTCU SX1 Series
RMONTGT_ICP109	109	RTCU AX9 Series
RMONTGT_ICP122	122	RTCU MX2 turbo Series
RMONTGT_ICP129	129	RTCU AX9 turbo / AX9 encore Series
RMONTGT_ICP192	192	RTCU MX2 SOM
RMONTGT_ICP202	202	RTCU NX-200 Series
RMONTGT_ICP204	204	RTCU NX-400
RMONTGT_ICP209	209	RTCU NX-900 Series
RMONTGT_ICP304	304	RTCU LX4 Series
RMONTGT_ICP305	305	RTCU LX5 Series.

rmonGetTargetProfile()

RTCU architecture: All
Called in: Connected State

Synopsis

rmonRet __stdcall rmonGetTargetProfile(HRMONCON hCon, int* targetID, int* firmwareVer, unsigned long* SerialNumber);

Description

Fetches RTCU type, serial number and firmware version from the RTCU.
On NX32L devices, the major and minor part of the runtime version is returned. (see rmonGetDeviceInfo())

Input

hCon	Handle to connection

Output

targetID	Please see the table below for a list of possible target ID's.
firmwareVer	Firmware version always different from 0 and scaled by 100 (Version 4.66 is returned as 466)
SerialNumber	The serialnumber of the RTCU device.

Returns

rmonOK, rmonComError, rmonIllegalHandle

Symbolic name	Value	Description
RMONTGT_ICP002	1	RTCU SA
RMONTGT_ICP003	2	RTCU DIN
RMONTGT_ICP004	4	RTCU D4
RMONTGT_ICP005	5	RTCU A5 / RTCU A5i
RMONTGT_ICP006	6	RTCU A6
RMONTGT_ICP007	7	RTCU M7
RMONTGT_ICP009	9	RTCU A9i
RMONTGT_ICP010	10	RTCU M10 Series
RMONTGT_ICP011	11	RTCU M11 Series
RMONTGT_ICP102	102	RTCU MX2 Series
RMONTGT_ICP103	103	RTCU DX4i mk2
RMONTGT_ICP104	104	RTCU DX4 Series
RMONTGT_ICP105	105	RTCU CX1 Series
RMONTGT_ICP106	106	RTCU SX1 Series
RMONTGT_ICP109	109	RTCU AX9 Series
RMONTGT_ICP122	122	RTCU MX2 turbo / RTCU MX2 encore
RMONTGT_ICP129	129	RTCU AX9 turbo / RTCU AX9 encore
RMONTGT_ICP192	192	RTCU MX2 SOM.
RMONTGT_ICP202	202	RTCU NX-200
RMONTGT_ICP204	204	RTCU NX-400
RMONTGT_ICP209	209	RTCU NX-900
RMONTGT_ICP304	304	RTCU LX4 Series
RMONTGT_ICP305	305	RTCU LX5 Series.

rmonGetDeviceInfo()

RTCU architecture: All
Called in: Connected State

Synopsis

rmonRet __stdcall rmonGetDeviceInfo(HRMONCON hCon, int* targetID, int* VerRuntime, int* VerSystem, unsigned long* SerialNumber);

Description

Fetches RTCU type, serial number and firmware version from the RTCU.

On NX32L devices, the version is <major>.<minor>.<build>; which is returned as a binary packed value using this format:

reserved	bit 24..31
major	bit 16..23
minor	bit 8..15
build	bit 0..7

On other RTCU devices the version is scaled by 100.(Version 4.66 is returned as 466)

Input

hCon	Handle to connection

Output

targetID	Please see the table below for a list of possible target ID's.
VerRuntime	The version of the runtime on the RTCU. This is always differen from 0.
VerSystem	The version of the system on the RTCU. This will be 0 on RTCU without a system version
SerialNumber	The serialnumber of the RTCU device.

Returns

rmonOK, rmonComError, rmonIllegalHandle

Symbolic name	Value	Description
RMONTGT_ICP002	1	RTCU SA
RMONTGT_ICP003	2	RTCU DIN
RMONTGT_ICP004	4	RTCU D4
RMONTGT_ICP005	5	RTCU A5 / RTCU A5i
RMONTGT_ICP006	6	RTCU A6
RMONTGT_ICP007	7	RTCU M7
RMONTGT_ICP009	9	RTCU A9i
RMONTGT_ICP010	10	RTCU M10 Series
RMONTGT_ICP011	11	RTCU M11 Series
RMONTGT_ICP102	102	RTCU MX2 Series
RMONTGT_ICP103	103	RTCU DX4i mk2
RMONTGT_ICP104	104	RTCU DX4 Series
RMONTGT_ICP105	105	RTCU CX1 Series
RMONTGT_ICP106	106	RTCU SX1 Series
RMONTGT_ICP109	109	RTCU AX9 Series
RMONTGT_ICP122	122	RTCU MX2 turbo / RTCU MX2 encore
RMONTGT_ICP129	129	RTCU AX9 turbo / RTCU AX9 encore
RMONTGT_ICP192	192	RTCU MX2 SOM.
RMONTGT_ICP202	202	RTCU NX-200
RMONTGT_ICP204	204	RTCU NX-400

rmonReceiveDebugMsg()

RTCU architecture: All
Called in: Connected State

Synopsis rmonRet __stdcall rmonReceiveDebugMsg (HRMONCON hCon, char* msg, int maxsize)

Description Receive any incoming Debug messages from RTCU.
Please notice that rmonReceiveDebugMsg blocks and will not return before a debug message has been received.

Input

hCon	Handle to connection
maxsize	Maximum number of characters to receive

Output

msg	Buffer with received debug message

Returns rmonOK, rmonIllegalHandle, rmonNoData

rmonGetDebugEnable()

RTCU architecture: All
Called in: Connected State

Synopsis rmonRet __stdcall rmonGetDebugEnable(HRMONCON hCon, int* enabled)

Description Checks if Debug messages has been enabled or disabled in device.

Input

hCon	Handle to connection

Output

enabled	1 if Debug messages is enabled, 0 if Debug messages is disabled

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

rmonSetDebugEnable()

RTCU architecture: All
Called in: Connected State

Synopsis rmonRet __stdcall rmonSetDebugEnable(HRMONCON hCon, int enabled)

Description Enable or disable Debug messages from device.

Input

hCon	Handle to connection
enabled	1 to enable Debug messages, 0 to disable Debug messages

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

rmonVoiceMessagesAbove64K()

RTCU architecture: X32
Called in: Connected State

Synopsis rmonRet __stdcall rmonVoiceMessagesAbove64K (HRMONCON hCon, char *above)

Description Determine if voice messages are stored above 64k.
This function is used to determine if voice messages will be overwritten by the rmonApplicationStartUpload function or the rmonFirmwareStartUpload function.

Input

hCon	Handle to connection

Output

above	1 if Voice messages above 64k, 0 if no Voice messages above 64k

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

rmonGetAppInfo()

RTCU architecture: All
Called in: Connected State

Synopsis rmonRet __stdcall rmonGetAppInfo(HRMONCON hCon, char *Appname, int *Appver)

Description Fetches Application name and version from the RTCU.

Input

hCon	Handle to connection

Output

Appname	0 (zero) terminated string containing the application name. Max. 15 characters long.
Appver	Application version scaled by 100 (Version 4.66 is returned as 466)

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError, rmonNoData

rmonGetGPRSSettings()

RTCU architecture: All
Called in: Connected State

Synopsis rmonRet __stdcall rmonGetGPRSSettings(HRMONCON hCon, rmonGPRSSettings* Settings);

Description This function fetches the TCP/IP settings the device uses to connect over GPRS.
The GPRS settings retrieved from the RTCU device are identical to those retrieved with the “Fetch” button in the RTCU IDE (Device->Network->Network settings).

All the general TCP/IP parameters use a binary packed IP address (a.b.c.d) using this format:

a	bit 24..31
b	bit 16..23
c	bit 8..15
d	bit 0..7

Input

hCon	Handle to connection

Output

Settings	A structure containing the GPRS settings

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError, rmonNoData

```
typedef struct {
  // general TCP/IP parameters:
  unsigned long  ip_address;
  unsigned long  subnet_mask;
  unsigned long  gateway;
  unsigned long  dns_1;
  unsigned long  dns_2;
  // PPP parameters:
  char           username[34];
  char           password[34];
  // Dialup/GPRS parameters:
  char           APN[34];
  unsigned short authentication;
} rmonGPRSSettings;
```

rmonSetGPRSSettings()

RTCU architecture: All
Called in: Connected State

Synopsis

rmonRet __stdcall rmonSetGPRSSettings(HRMONCON hCon, rmonGPRSSettings Settings);

Description

This function sets the TCP/IP settings the device uses to connect over GPRS.
This function is identical to the VPL function sockSetTCPIPParm, and the TCP/IP settings dialog (Device->Network->Network settings) in the RTCU IDE.

All the general TCP/IP parameters use a binary packed IP address (a.b.c.d) using this format:

a	bit 24..31
b	bit 16..23
c	bit 8..15
d	bit 0..7

Input

hCon	Handle to connection
Settings	A structure containing the GPRS settings

Returns

rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

```
typedef struct {
  // general TCP/IP parameters:
  unsigned long  ip_address;
  unsigned long  subnet_mask;
  unsigned long  gateway;
  unsigned long  dns_1;
  unsigned long  dns_2;
  // PPP parameters:
  char           username[34];
  char           password[34];
  // Dialup/GPRS parameters:
  char           APN[34];
  unsigned short authentication;
} rmonGPRSSettings;
```

rmonGetGatewaySettings()

RTCU architecture: All
Called in: Connected State

Synopsis

rmonRet __stdcall rmonGetGatewaySettings(HRMONCON hCon, rmonGWSettings* Settings);

Description

This function fetches the settings the device uses to connect to the RCH.
The RCH settings retrieved from the RTCU device are identical to those retrieved in the RTCU IDE with the “Fetch” button in the RCH settings dialog (Device->Network->RTCU Communication Hub settings).

Input

hCon	Handle to connection

Output

Settings	A structure containing the RCH settings

Returns

rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

```
typedef struct {
  // RTCU Communication Hub parameters:
  unsigned short gw_enabled;
  char           gw_ip[42];
  unsigned short gw_port;
  char           gw_key[10];
  char           phonenumber_sms[22];
  unsigned char  crypt_key[16];
  // advanced settings (modification not recommended):
  unsigned short max_connection_attempt;
  unsigned short max_send_req_attempt;
  unsigned short response_timeout;
  unsigned short alive_freq;
} rmonGWSettings;
```

rmonSetGatewaySettings()

RTCU architecture: All
Called in: Connected State

Synopsis

rmonRet __stdcall rmonSetGatewaySettings(HRMONCON hCon, rmonGWSettings Settings);

Description

This function fetches the settings the device uses to connect to RCH.
This function is identical to the VPL function sockSetGWParm, and the settings are identical to those written from the RTCU IDE with the “Apply” button in the RTCU Communication Hub settings dialog (Device->Network->RTCU Communication Hub settings).

Input

hCon	Handle to connection
Settings	A structure containing the Gateway settings

Returns

rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

```
typedef struct {
  // RTCU Communication Hub parameters:
  unsigned short gw_enabled;
  char           gw_ip[42];
  unsigned short gw_port;
  char           gw_key[10];
  char           phonenumber_sms[22];
  unsigned char  crypt_key[16];
  // advanced settings (modification not recommended):
  unsigned short max_connection_attempt;
  unsigned short max_send_req_attempt;
  unsigned short response_timeout;
  unsigned short alive_freq;
} rmonGWSettings;
```

rmonRCHGetConfig()

RTCU architecture: NX32L
Called in: Connected State

Synopsis

rmonRet __stdcall rmonRCHGetConfig(HRMONCON hCon, const int index, rmonRCHConfig* Settings);

Description

This function fetches the RTCU Communication Hub settings the device uses to connect to server.
The RCH settings retrieved from the RTCU device are identical to those retrieved in the RTCU IDE with the “Fetch” button in the RTCU Communication Hub settings dialog (Device→Network→RTCU Communication Hub settings).

Input

hCon	Handle to connection
index	The index of the configuration to read. 0 = Fallback config, 1 = Primary server config.

Output

Settings	A structure containing the settings

Returns

rmonOK, rmonErrorr, rmonComError, rmonIllegalHandle, rmonTargetError

```
struct rmonRCHConfig {
  unsigned long  size;
  unsigned char  enabled;
  unsigned char  secure;
  char           host[51];
  unsigned char  iface;
  unsigned char  crypt_enable;
  unsigned short port;
  char           login_key[10];
  unsigned char  crypt_key[16];
  unsigned short max_connection_attempt;
  unsigned short max_send_req_attempt;
  unsigned short response_timeout;
  unsigned short alive_freq;
};
```

rmonRCHSetConfig()

RTCU architecture: NX32L
Called in: Connected State

Synopsis

rmonRet __stdcall rmonRCHSetConfig(HRMONCON hCon, const int index, rmonGWSettings Settings);

Description

This function fetches the RTCU Communication Hub settings the device uses to connect to the server.
This function is identical to the VPL function rchConfigSet, and the settings are identical to those written from the RTCU IDE with the "Apply" button in the RTCU Communication Hub settings dialog (Device→Network→RTCU Communication Hub settings).

Input

hCon	Handle to connection
index	The index of the configuration to read. 0 = Fallback config, 1 = Primary server config.
Settings	A structure containing the RCH settings

Returns

rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

```
struct rmonGWSettings {
  unsigned long  size;
  unsigned char  enabled;
  unsigned char  secure;
  char           host[51];
  unsigned char  iface;
  unsigned char  crypt_enable;
  unsigned short port;
  char           login_key[10];
  unsigned char  crypt_key[16];
  unsigned short max_connection_attempt;
  unsigned short max_send_req_attempt;
  unsigned short response_timeout;
  unsigned short alive_freq;
};
```

rmonRCHGetAutoConnect()

RTCU architecture: NX32L
Called in: Connected State

Synopsis

rmonRet __stdcall rmonRCHGetAutoConnect(HRMONCON hCon, int *enable)

Description

This function fetches whether the RTCU device will start the connection to the RTCU Communication Hub automatically, or requires the application to do this manually.

Input

hCon	Handle to connection

Output

enable	1 if auto connect is enabled, 0 if auto connect is disabled

Returns

rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError, rmonNoData

rmonRCHSetAutoConnect()

RTCU architecture: NX32L
Called in: Connected State

Synopsis
rmonRet __stdcall rmonRCHGetAutoConnect(HRMONCON hCon, int *enable)

Description
This function sets whether the RTCU device will start the connection to the RTCU Communication Hub automatically, or requires the application to do this manually.

Input

hCon	Handle to connection
enable	1 if auto connect is enabled, 0 if auto connect is disabled

Returns
rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError, rmonNoData

rmonGetLANSettings()

RTCU architecture: NX32 & NX32L
Called in: Connected State

Synopsis
rmonRet __stdcall rmonGetLANSettings(HRMONCON hCon, int iface, rmonNetwork* Settings);

Description
This function fetches the settings the device uses to connect over Ethernet.
The LAN settings retrieved from the RTCU device are identical to those retrieved in the RTCU IDE with the “Fetch” button in the Network settings dialog (Device->Network->Network settings).

Input

hCon	Handle to connection
iface	The network interface

Output

Settings	A structure containing the LAN settings

Returns
rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

```
typedef struct {
  unsigned short flags;
  unsigned long  addr;
  unsigned long  subnet;
  unsigned long  gateway;
  unsigned long  dns_1;
  unsigned long  dns_2;
} rmonNetwork;
```

Available flags:

Symbolic name	Value	Description
RMON_NET_DHCP	0x0001	Use DHCP to get TCP/IP address.
RMON_NET_AUTODNS	0x0002	Use DNS servers from DHCP lookup.

rmonSetLANSettings()

RTCU architecture: NX32 & NX32L
Called in: Connected State

Synopsis

rmonRet __stdcall rmonSetLANSettings(HRMONCON hCon, int iface, rmonNetwork Settings);

Description

This function sets the settings the device uses to connect over Ethernet.
This function is identical to the VPL function netSetLANParam, and the settings are identical to those written from the RTCU IDE with the “Apply” button in the Network settings dialog (Device->Network->Network settings).

Input

hCon	Handle to connection
iface	The network interface
Settings	A structure containing the LAN settings

Output

Returns

rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

```
typedef struct {
  unsigned short flags;
  unsigned long  addr;
  unsigned long  subnet;
  unsigned long  gateway;
  unsigned long  dns_1;
  unsigned long  dns_2;
} rmonNetwork;
```

Available flags:

Symbolic name	Value	Description
RMON_NET_DHCP	0x0001	Use DHCP to get TCP/IP address.
RMON_NET_AUTODNS	0x0002	Use DNS servers from DHCP lookup.

rmonGetWLANSettings()

RTCU architecture: NX32L
Called in: Connected State

Synopsis

rmonRet __stdcall rmonGetWLANSettings(HRMONCON hCon, int index, rmonWLANSettings* Settings);

Description

This function fetches the settings the device uses to connect over WiFi.
The WLAN settings retrieved from the RTCU device are identical to those retrieved in the RTCU IDE with the “Fetch” button in the Network settings dialog (Device->Network->Network settings).

Input

hCon	Handle to connection
index	The index of the wireless network information

Output

Settings	A structure containing the WLAN settings

Returns

rmonOK, rmonComError, rmonError, rmonIllegalHandle, rmonTargetError

```
typedef struct {
  unsigned short flags;
  unsigned long  addr;
  unsigned long  subnet;
  unsigned long  gateway;
  unsigned long  dns_1;
  unsigned long  dns_2;
} rmonNetwork;

typedef struct {
  char           phrase[64];
} rmonWLANSecurityWPA;

typedef struct {
  unsigned char  ssid[32];
  unsigned short security;
  union {
    rmonWLANSecurityWPA wpa;
  } sec;
  rmonNetwork    tcpip;
} rmonWLANSettings;
```

Available flags:

Symbolic name	Value	Description
RMON_NET_DHCP	0x0001	Use DHCP to get TCP/IP address.
RMON_NET_AUTODNS	0x0002	Use DNS servers from DHCP lookup.

rmonSetWLANSettings()

RTCU architecture: NX32L
Called in: Connected State

Synopsis

rmonRet __stdcall rmonSetWLANSettings(HRMONCON hCon, int index, rmonWLANSettings Settings);

Description

This function sets the settings the device uses to connect over WiFi.
This function is identical to the VPL function netSetWLANParam, and the settings are identical to those written from the RTCU IDE with the “Apply” button in the Network settings dialog (Device->Network->Network settings).

Input

hCon	Handle to connection
index	The index of the wireless network information
Settings	A structure containing the WLAN settings

Returns

rmonOK, rmonComError, rmonError, rmonIllegalHandle, rmonTargetError

```
typedef struct {
  unsigned short flags;
  unsigned long  addr;
  unsigned long  subnet;
  unsigned long  gateway;
  unsigned long  dns_1;
  unsigned long  dns_2;
} rmonNetwork;

typedef struct {
  char           phrase[64];
} rmonWLANSecurityWPA;

typedef struct {
  unsigned char  ssid[32];
  unsigned short security;
  union {
    rmonWLANSecurityWPA wpa;
  } sec;
  rmonNetwork    tcpip;
} rmonWLANSettings;
```

Available flags:

Symbolic name	Value	Description
RMON_NET_DHCP	0x0001	Use DHCP to get TCP/IP address.
RMON_NET_AUTODNS	0x0002	Use DNS servers from DHCP lookup.

rmonFaultLogRead()

RTCU architecture: All
Called in: Connected State

Synopsis rmonRet __stdcall rmonFaultLogRead(HRMONCON hCon, rmonFault *Fault)

Description Fetches the Fault log from the RTCU.
This function is identical to the fetch button in the fault log in the RTCU IDE

Input

hCon	Handle to connection

Output

Fault	The function fills this structure with the fault log entries.

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

```
typedef struct {
  unsigned short   year;
  unsigned char    month;
  unsigned char    date;
  unsigned char    hour;
  unsigned char    minute;
  unsigned char    second;
  unsigned char    Code;
} rmonFaultRecord;

typedef struct {
  unsigned char    NumRecords;
  unsigned char    NextIn; // Index where the next record will be
inserted
  rmonFaultRecord  Record[32];
} rmonFault;
```

rmonFaultLogReadX()

RTCU architecture: All
Called in: Connected State

Synopsis rmonRet __stdcall rmonFaultLogRead(HRMONCON hCon, rmonFaultX *Fault)

Description Fetches the Fault log from the RTCU, with debug information.
This function is identical to the fetch button in the fault log in the RTCU IDE

Input

hCon	Handle to connection

Output

Fault	The function fills this structure with the fault log entries.

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

```
typedef struct {
  unsigned short   year;
  unsigned char    month;
  unsigned char    date;
  unsigned char    hour;
  unsigned char    minute;
  unsigned char    second;
  unsigned char    code;
  char             filename[16];
  unsigned short   line;
  unsigned short   threadid;
  unsigned char    data[38];
} rmonFaultRecordX;

typedef struct {
  unsigned char    NumRecords;
  unsigned char    NextIn; // Index where the next record will be
inserted
  rmonFaultRecordX Record[16];
} rmonFaultX;
```

rmonFaultLogClear()

RTCU architecture: All
Called in: Connected State

Synopsis rmonRet __stdcall rmonFaultLogClear(HRMONCON hCon)

Description This function clears the fault log.
This function is identical to the clear button in the Fault log in the RTCU IDE.

Input

hCon	Handle to connection

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

rmonFaultGetText()

RTCU architecture: All
Called in: Connected State

Synopsis rmonRet __stdcall rmonFaultGetText(unsigned char fault, char* FaultText, int bufsize)

Description This function retrieves the text message for a Fault code

Input

fault	The fault code
bufsize	The size of the buffer where the text is to be stored. If bufsize exceeds the number of characters in the fault text, only the number of characters present will be put in the output buffer.

Output

FaultText	0 (zero) terminated string containing the fault message

Returns rmonOK

rmonSoftwareUpgrade()

RTCU architecture: All
Called in: Connected State

Synopsis rmonRet __stdcall rmonSoftwareUpgrade(HRMONCON hCon, char string[35], int *res)

Description Upgrades the RTCU device.

Input

hCon	Handle to connection
String	0 (zero) terminated string. The upgrade key.

Output

res	The type of upgrade performed.

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError, rmonError, rmonIllegalTarget

Value	Description
0	Not upgraded / Wrong upgrade key
1	GPRS enabled
2	Device is programmable
3	LCD display enabled
4	Clear password.
5	Battery enabled.
6	FMI support enabled.
7 - 9	Not used
10	Citect SCADA enabled
11	Web enabled

rmonFlexOption()

RTCU architecture: All
Called in: Connected State

Synopsis rmonRet __stdcall rmonFlexOptiont(HRMONCON hCon, char string[12])

Description This function will enable certain options in the device.

Input

hCon	Handle to connection
string	Zero terminated string. The option key.

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError, rmonError

rmonStatisticsRead()

RTCU architecture: All
Called in: Connected State

Synopsis rmonRet __stdcall rmonStatisticsRead(HRMONCON hCon, rmonUnitStatistics *data)

Description Fetches the device statistics from the RTCU.
The size parameter in the structure must be set to the size of the structure (sizeof) before this function is called.

Input

hCon	Handle to connection

Output

data	The function fills this structure with the statistics.

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonTargetError

```
typedef struct {
  short             size;
  short             max_temp;
  short             avg_temp;
  short             min_temp;
  unsigned long     num_boot;
  unsigned long     time_run;
  unsigned long     time_bat;
  unsigned long     time_chg;
  unsigned long     gw_connect;
  unsigned long     gw_incoming;
  unsigned long     gw_outgoing;
  unsigned long     gw_alive;
  unsigned long     gprs_connect;
  unsigned long     gprs_send;
  unsigned long     gprs_receive;
  unsigned long     gps_nofix;
  unsigned long     gps_2dfix;
  unsigned long     gps_3dfix;
} rmonUnitStatistics;
```

The structure variables hold the following statistics:

size	**The size of the structure.**
max_temp	The highest temperature measured in the device, represented in 0.01 deg. Celsius.
avg_temp	The average temperature measured in the device, represented in 0.01 deg. Celsius.
min_temp	The lowest temperature measured in the device, represented in 0.01 deg. Celsius.
num_boot	The number of times the device has rebooted.
time_run	The number of minutes the device has been operating.
time_bat	The number of minutes the device has been operating from the backup battery.
time_chg	The number of minutes the device have been charging the backup battery.
gw_connect	The number of times the device has tried to connect to RCH.
gw_incoming	The number of incoming transactions from the RCH.
gw_outgoing	The number of outgoing transactions to the RCH.
gw_alive	The number of keep-alive transactions the device has sent to the RCH.
gprs_connect	The number of times the device successfully connected to GPRS.
gprs_send	The number of bytes send over the GPRS connection.
gprs_receive	The number of bytes received from the GPRS connection.
gps_nofix	The number of times the GPS module has reported 'No fix'.

gps_2dfix	The number of times the GPS module has reported '2D fix'.
gps_3dfix	The number of times the GPS module has reported '3D fix'.

rmonGetUnitState()

RTCU architecture: All
Called in: Connected State

Synopsis rmonRet __stdcall rmonGetUnitState(HRMONCON hCon, rmoncbunitstate pfunc, void* uptr)

Description Read the execution state from the RTCU device and return it in a call-back function

Input

hCon	Handle to connection
pfunc	Function that is called with the name and description of a serial port.
uptr	A user defined argument that is included in the callback function.

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonError

The call-back function is defined as follows:

typedef void (__stdcall *rmoncbunitstate)(void* uptr, int state);
state:

Symbolic name	Value	Description
RMON_STATE_RUN	1	RTCU is running
RMON_STATE_HALT	2	RTCU is halted
RMON_STATE_FAULT	3	RTCU is faulted

rmonGetPowerInformation()

RTCU architecture: All
Called in: Connected State

Synopsis rmonRet __stdcall rmonGetPowerInformation(HRMONCON hCon, rmon_power_info_t *data)

Description Read the current environment from RTCU device.

Input

hCon	Handle to connection

Output

data	The function fills this structure with the environment data

Returns rmonOK, rmonComError, rmonIllegalHandle, rmonError

```
typedef struct {
  short            sof;
  short            temp;
  unsigned short   sup_volt;
  unsigned char    sup_type;
  unsigned char    batt_type;
  unsigned char    batt_level;
  unsigned char    batt_charging;
  unsigned char    chg_enable;
  unsigned char    run_on_batt;
  char             ext_disable;
  char             s0_enable;
  char             dcout1_enable;
  char             dcout2_enable;
  char             lcd_power;
} rmon_power_info_t;
```

The structure variables hold the following information:

sof	**The size of the structure. The size must be set before calling the function.**
temp	The temperature in the device. (scaled by 100)
sup_volt	The power supply voltage. (scaled by 10)
sup_type	The power supply type.
batt_type	The type of battery present.
batt_level	The battery power level. (0..5)
batt_charging	The battery is being charged.
chg_enable	The battery charger is enabled.
run_on_batt	The device will continue to run from the battery if the supply is removed.
ext_disable	The external power is disabled.
s0_enable	S0 mode is enabled for digital inputs.
dcout1_enable	The DC-OUT 1 is enabled.
cdout2_enable	The DC-OUT 2 is enabled.
LCD_power	The display is powered.

Supply type:

Symbolic name	Value	Description
RMON_SUPPLY_BAT	1	Operating on internal battery.
RMON_SUPPLY_DC	2	Operating on external DC power.
RMON_SUPPLY_AC	3	Operating on external AC power.

Battery type:

Symbolic name	Value	Description
RMON_BAT_NONE	0	No battery.
RMON_BAT_NONCHG	1	Non-chargeable battery
RMON_BAT_LOW	2	Low capacity battery
RMON_BAT_HIGH	3	High capacity battery
RMON_BAT_SUPER	4	Super capacity battery

The enable or disable type:

Value	Description
-1	Not available
0	Disabled
1	Enabled

Appendix A, simple application

```
//-------------------------------------------------------------
// Small RTCUCSP.DLL sample program
//-------------------------------------------------------------
#include <windows.h>
#include <process.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <rtcucsp.h>

//-------------------------------------------------------------
// Connection handle
//-------------------------------------------------------------
HRMONCON hCon;

//-------------------------------------------------------------
// Receive any incoming Debug messages from RTCU
//-------------------------------------------------------------
static void thDebug(void *arg) {
  char buffer[512];
  int rc;
  for (;;) {
    // Wait for any debug messages from unit
    rc=rmonReceiveDebugMsg(hCon, buffer, sizeof(buffer));
    if (rc==0) {
      // Just print the debug message
      printf("Debug::[%s]\n", buffer);
    } else {
      Sleep(50);
    }
  }
}

//-------------------------------------------------------------
// Callback function that will be called by rmonFirmwareUpload()
// to report progress in the upload process
//-------------------------------------------------------------
static int RMONCC cbfuncFW(void* uptr,int percentage) {
  printf("Firmware upload finished: %3i\r", percentage);
  return 0;
}

//-------------------------------------------------------------
// Callback function that will be called by rmonApplicationUpload()
// to report progress in the upload process
//-------------------------------------------------------------
static int RMONCC cbfuncApp(void* uptr,int percentage) {
  printf("Application upload finished: %3i\r", percentage);
  return 0;
}

//-------------------------------------------------------------
// Callback function that will be called by rmonVoiceUpload()
// to report progress in the upload process
//-------------------------------------------------------------
static int RMONCC cbfuncVoice(void* uptr,int percentage) {
  printf("Voice upload finished: %3i\r", percentage);
  return 0;
}

//-------------------------------------------------------------
// the main program
//-------------------------------------------------------------
int main(int argc,char** argv) {
  int rc;

  // Open RTCUCSP library
  rmonOpen();
```

```
  // Open a connection
  hCon = rmonOpenConnection();

  // Select which comports to use for local and remote connections. (Use COM0 if no port is to be
used)
  rmonSetComport(hCon, "COM1", "COM0");

  // Connect to unit via cable
  rmonConnect(hCon, "");

  // Start the listener thread for incoming Debug messages
  _beginthread(thDebug, 0, NULL);

  // Wait for a connection to a RTCU unit
  while (1) {
    // Check and wait for connection (can be RMONCON_LOCAL, RMONCON_REMOTE, RMONCON_GW or
RMONCON_NONE)
    if (rmonConnected(hCon) != RMONCON_NONE)
      break;
    Sleep(300);
  }
  printf("Connected to unit.\n");

  // Try to authenticate with an empty password:
  rc=rmonAuthenticate(hCon, "");
  switch (rc) {
    case rmonDenied: printf("Authenticate with empty password denied. Use correct password !\n");
break;
    case rmonOK: printf("Authenticate with empty password accepted !\n"); break;
  }

  if (rc==rmonDenied) {
    printf("Not able to logon to unit !\n");
    return 1;
  }

  // Get information about the unit
  int targetid, firmwareversion;
  rmonGetTargetInfo(hCon, &targetid, &firmwareversion);
  printf("Target ID=%i, Firmware version=%i.%02i\n", targetid, firmwareversion/100,
firmwareversion%100);

  // Get the units serial number
  unsigned long SerialNumber;
  rmonGetSerialNumber(hCon, &SerialNumber);
  printf("Serialnumber of unit is %09i\n", SerialNumber);


  // Use this to upload new firmware to the unit:
  //printf("\nrc=%i\n", rmonFirmwareUpload(hCon, "D:\\FirmwareFile.bin", cbfuncFW, (void*)0));


  // Use the following to upload a new application and voice messages:
  /*
  rmonHalt(hCon);
  printf("\nrc=%i\n", rmonApplicationUpload(hCon, "D:\\APP\\APP.VSX", cbfuncApp, NULL));
  printf("\nrc=%i\n", rmonVoiceUpload(hCon, "C:\\APP\\APP.PRJ", cbfuncVoice, NULL));
  rmonReset(hCon);
  */

  // Close connection after use
  rmonCloseConnection(hCon);

  printf("\n\nPress any key to end program...\n\n");
  while (1) {
    if (getch())
      break;
  }

 return 0;
}
```

Appendix B, RTCUPROG application

The RTCUPROG program is a complete Microsoft Visual Studio C++ 2019 project. This program demonstrates all aspects in making a robust application that will manage the connection to both local and remote RTCU device. The application allows the user to upload new firmware to a device, or upload a complete new project to the RTCU device.